

به نام خدا

نوشین نوذری

گزارش پروژه پایانی درس پردازش موازی

تابستان ۱۴۰۳

کارشناسی ارشد مهندسی نرم افزار

Defining a thread:

```
1 from fastapi import FastAPI, Query
2 import threading
3 output=[]
4 app=FastAPI()
5 # usage
6
7 def myfunc(thread_number:int):
8     message=f"my func called by thread N^{thread_number}"
9     output.append(message)
10
11 @app.get("/run1/")
12 def query(thread_count:int =Query(...,title="Number of threads",ge=1,le=10)):
13     global output
14     output=[]
15     threads=[]
16     for i in range(thread_count):
17         thread=threading.Thread(target=myfunc,args=(i,))
18         threads.append(thread)
19         thread.start()
20     for thread in threads:
21         thread.join()
22     return output
```

(۱)وارد کردن کتابخانه ها:

Fastapi:برای ایجاد و مدیریت API وب

Query:برای تنظیم پارامترها در قالب کوئری

Threading:برای مدیریت و ایجاد نخ های همزمان

(۲)تعریف متغیرها و APP

Output:لیستی از خروجی نخ ها را ذخیره میکند.

App:شی fastapi برای تعریف مسیرها و مدیریت درخواست ها

(۳)تعریف تابع نخ ها:

Myfunc:تابعی که توسط هر نخ فراخوانی میشود این تابع شماره هر نخ را به عنوان ورودی میگیرد و پیامی به لیست خروجی اضافه میکند.

(۴)تعریف مسیر API:

مسیر /run1/ این مسیر یک درخواست GET را پردازش می کند.

پارامتر **thread_count**: تعداد نخ هایی که باید ایجاد شوند. با استفاده از **Query** اعتبارسنجی شده است که باید بین ۱ و ۱۰ باشد.

output قبل از هر اجرای جدید خالی می شود تا نتایج قبلی پاک شوند.

threads: لیستی برای نگهداری نخ های ایجاد شده.

حلقه اول: نخ ها را ایجاد و شروع می کند.

حلقه دوم: منتظر می ماند تا تمامی نخ ها به پایان برسند (**join**).

return output: لیست **output** را به عنوان پاسخ برمی گرداند.

این کد به شما امکان می دهد تا با فراخوانی مسیر **/run1/?thread_count=10** و ارائه تعداد تردها به عنوان پارامتر، چندین ترد را به صورت همزمان اجرا کنید و خروجی های آنها را مشاهده کنید.

خروجی:

GET http://127.0.0.1:8000/run1



http://127.0.0.1:8000/run1/?thread_count=10

GET



http://127.0.0.1:8000/run1/?thread_count=10

Params ● Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

	Key	Value
<input checked="" type="checkbox"/>	thread_count	10
	Key	Value



Body Cookies Headers (4) Test Results

Pretty

Raw

Preview

Visualize

JSON



```
1
2 "my func called by thread N^0",
3 "my func called by thread N^1",
4 "my func called by thread N^2",
5 "my func called by thread N^3",
6 "my func called by thread N^4",
7 "my func called by thread N^5",
8 "my func called by thread N^6",
9 "my func called by thread N^7",
10 "my func called by thread N^8",
11 "my func called by thread N^9"
12
```

2) Determining the current thread

```

#2) Determining the current thread
import threading
import time
from typing import List
from fastapi import FastAPI, Query
from pydantic import BaseModel
app = FastAPI()

3 usages
class ResponseModel(BaseModel):
    message: str

1 usage
def function_A():
    time.sleep(1) # simulate some processing
    return "function_A--> starting", "function_A--> exiting"

1 usage
def function_B():
    time.sleep(2) # simulate some processing
    return "function_B--> starting", "function_B--> exiting"

1 usage
def function_C():
    time.sleep(3) # simulate some processing
    return "function_C--> starting", "function_C--> exiting"

1 usage
def run_functions(thread_count: int) -> List[str]:
    threads = []
    results = []
    functions = [function_A, function_B, function_C]

```

```

def run_functions(thread_count: int) -> List[str]:
    threads = []
    results = []
    functions = [function_A, function_B, function_C]

    if thread_count > len(functions):
        thread_count = len(functions)

    for i in range(thread_count):
        thread = threading.Thread(target=lambda func: results.extend(func()), args=(functions[i],))
        threads.append(thread)
        thread.start()

    for thread in threads:
        thread.join()

    return results

@app.get("/run2/", response_model=List[ResponseModel])
def execute_functions(thread_count: int = Query(..., title="Number of threads", ge=1, le=3)) -> List[ResponseModel]:
    results = run_functions(thread_count)
    response_models = [ResponseModel(message=msg) for msg in results]
    return response_models

```

(۱) استفاده از threading توابع موازی

Threading برای ایجاد و مدیریت نخ های موازی استفاده میشود و تایم برای ایجاد یک تاخیر مصنوعی در هر تابع استفاده میشود.

(۲) تعریف مدل داده

در fastapi لازم است داده از قبل تعریف شده و نوع آن مشخص شود

(۳) تعریف توابع موازی

این بخش شامل تعریف سه تابع است که هر کدام یک پردازش ساده را شبیه سازی میکند هر تابع با استفاده از تایم اسلیپ یک تاخیر مصنوعی دادرد تا واضح مدت زمان هر کدام مشخص شود این توابع پیامی برای نشان دادن شروع و خاتمه هر تابع برمیگرداند. در این بخش، run_functions تعداد مورد نظری از توابع موازی را به عنوان ورودی دریافت می کند و آن ها را با استفاده از threading.Thread در نخ های جداگانه اجرا می کند. پس از اتمام اجرای هر تابع، نتایج به results اضافه می شوند و در نهایت، تمامی نخ ها با استفاده از thread.join() منتظر می مانند تا اجرای همه توابع تمام شود

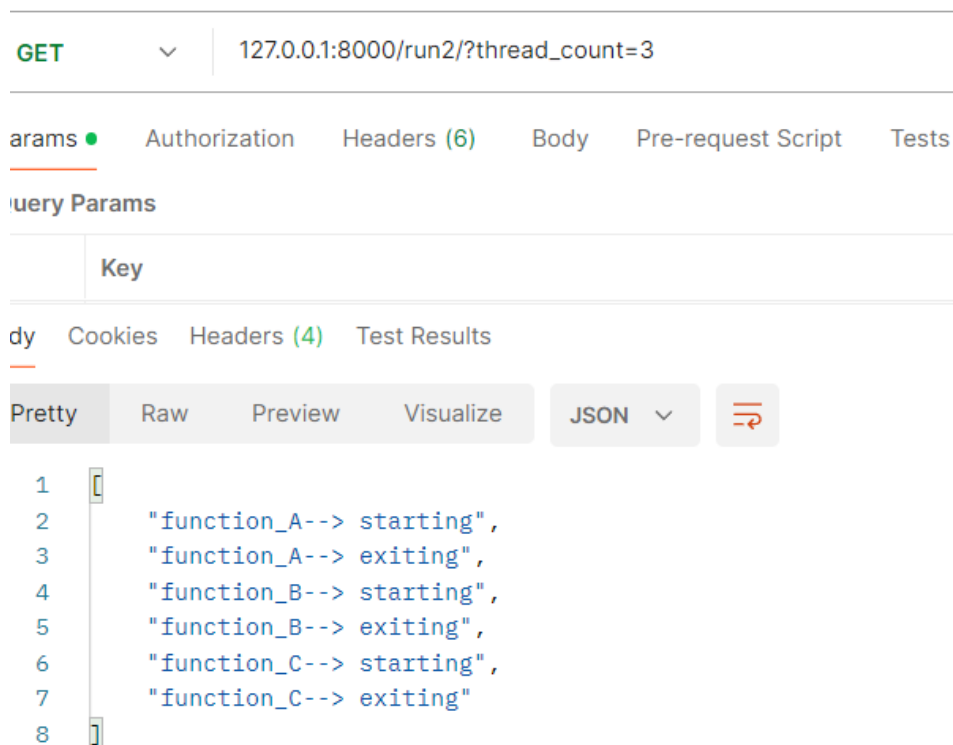
(۴) تعریف API اصلی با fastapi

در این بخش، یک نمونه از FastAPI ساخته می شود و یک مسیر /run2/ برای API تعریف می شود. این API یک پارامتر thread_count را از کاربر دریافت می کند که تعداد نخ هایی که باید برای اجرای توابع موازی استفاده شود را

مشخص می‌کند. در پایان، نتایج به صورت لیستی از `ResponseModel` تبدیل شده و به عنوان خروجی API ارسال می‌شوند.

این توضیحات به شما کمک می‌کنند تا هر بخش از کد را درک کنید و بفهمید که چگونه توابع موازی با استفاده از `threading` در یک برنامه `FastAPI` پیاده‌سازی شده‌اند.

خروجی:



```
from typing import List
from fastapi import FastAPI, Query
from pydantic import BaseModel

app = FastAPI()

3 usages
class ResponseModel(BaseModel):
    message: str

3 usages
def function_A() -> List[str]:
    return ["function_A--> starting", "function_A--> exiting"]

3 usages
def function_B() -> List[str]:
    return ["function_B--> starting", "function_B--> exiting"]

3 usages
def function_C() -> List[str]:
    return ["function_C--> starting", "function_C--> exiting"]

1 usage
```



```

1 usage
def run_functions() -> List[str]:
    results = []
    results.append(function_A()[0])
    results.append(function_B()[0])
    results.append(function_C()[0])
    results.append(function_A()[1])
    results.append(function_B()[1])
    results.append(function_C()[1])
    return results

@app.get("/run111/", response_model=List[ResponseModel])
def execute_functions() -> List[ResponseModel]:
    results = run_functions()
    response_models = [ResponseModel(message=msg) for msg in results]
    return response_models

```

در این بخش از کد، توابع `function_A`، `function_B` و `function_C` صدا زده می‌شوند و خروجی آن‌ها به ترتیب به لیست `results` اضافه می‌شود. این توابع به ترتیب دو عنصر در لیست برمی‌گردانند:

۱. `function_A()[0]`، `function_B()[0]`، `function_C()[0]`: این بخش از کد، پیام شروع از هر یک از توابع `function_A`، `function_B` و `function_C` را به لیست `results` اضافه می‌کند. به عبارت دیگر، اولین عنصر بازگشتی هر یک از این توابع که نماینده‌ی پیام شروع است به لیست اضافه می‌شود.

۲. `function_A()[1]`، `function_B()[1]`، `function_C()[1]`: این بخش از کد، پیام خاتمه از هر یک از توابع `function_A`، `function_B` و `function_C` را به لیست `results` اضافه می‌کند. به عبارت دیگر، دومین عنصر بازگشتی هر یک از این توابع که نماینده‌ی پیام خاتمه است به لیست اضافه می‌شود.

بنابراین، با این کد تضمین می‌شود که پیام‌های شروع و خاتمه از هر یک از توابع `function_A`، `function_B` و `function_C` به ترتیب صحیح به لیست `results` اضافه و برگشت داده شوند.

خروجی:

HTTP 127.0.0.1:8000/run111/?thread_count=3

GET

127.0.0.1:8000/run22/?thread_count=3

Params ● Authorization Headers (6) Body Pre-request Script Tests

Query Params

Key

Body Cookies Headers (4) Test Results

Pretty

Raw

Preview

Visualize

JSON



```
1  [
2    {
3      "message": "function_A--> starting"
4    },
5    {
6      "message": "function_B--> starting"
7    },
8    {
9      "message": "function_C--> starting"
10   },
11   {
12     "message": "function_A--> exiting"
13   },
14   {
15     "message": "function_B--> exiting"
16   },
17   {
18     "message": "function_C--> exiting"
19   }
20 ]
```

3) Defining a thread subclass

```
#3) Defining a thread subclass
from fastapi import FastAPI, Query
import threading
import time
import os
from typing import List
```

```
app = FastAPI()
```

```
output = []
```

```
1 usage
```

```
class MyThread(threading.Thread):
```

```
    def __init__(self, thread_number: int):
```

```
        super().__init__()
```

```
        self.thread_number = thread_number
```

```
    def run(self):
```

```
        global output
```

```
        pid = os.getpid()
```

```
        output.append(f"---> Thread#{self.thread_number} running, belonging to process ID {pid}")
```

```
        time.sleep(1)
```

```
        output.append(f"---> Thread#{self.thread_number} over")
```

```

@app.get("/run3/", response_model=List[str])
def run_threads(thread_count: int = Query(..., title="Number of threads", ge=1, le=10)) -> List[str]:
    global output
    output = []
    threads = []

    start_time = time.time()

    for i in range(thread_count):
        thread = MyThread(thread_number=i + 1)
        threads.append(thread)
        thread.start()

    for thread in threads:
        thread.join()

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

    output.append(f"End\n--- {total_time} seconds ---")

    return output

```

توضیح کد:

وارد کردن کتابخانه های `fastapi`, `threading`, `time`, `os`, `list`

و ایجاد یک شی از `FastAPI` و تعریف یک متغیر سراسری `output`

تعریف کلاس `My threads` و `__init__ method`: این متد سازنده کلاس است که شماره رشته (`thread number`) را به عنوان ورودی می پذیرد و آن را به یک متغیر نمونه اختصاص می دهد

`run method`. این متد زمانی که رشته شروع به اجرا می کند، فراخوانی می شود. در این متد:

شناسه پردازش فعلی با استفاده از `os.getpid()` به دست می آید.

پیامی به متغیر `output` اضافه می شود که نشان می دهد رشته در حال اجرا است

در انهایت با ارث بری از کلاس اولیه و ایجاد یک زیر کلاس به مقدار ۱۰ عدد نخ را همزمان اجرا میکند و با شماره آنها به خروجی میبرد در ابتدا آغاز را نمایش داده و نخ ها یکی یکی پایان میابد و در نهایت مدت زمان اجرای کل را به نمایش میگذارد

خروجی:

```

1  [
2  "----> Thread#1 running, belonging to process ID 14944",
3  "----> Thread#2 running, belonging to process ID 14944",
4  "----> Thread#3 running, belonging to process ID 14944",
5  "----> Thread#4 running, belonging to process ID 14944",
6  "----> Thread#5 running, belonging to process ID 14944",
7  "----> Thread#6 running, belonging to process ID 14944",
8  "----> Thread#7 running, belonging to process ID 14944",
9  "----> Thread#8 running, belonging to process ID 14944",
10 "----> Thread#9 running, belonging to process ID 14944",
11 "----> Thread#10 running, belonging to process ID 14944",
12 "----> Thread#3 over",
13 "----> Thread#1 over",
14 "----> Thread#2 over",
15 "----> Thread#9 over",
16 "----> Thread#8 over",
17 "----> Thread#10 over",
18 "----> Thread#7 over",
19 "----> Thread#6 over",
20 "----> Thread#5 over",
21 "----> Thread#4 over",
22 "End\n--- 1.0161736011505127 seconds ---"
3  ]

```

در سناریو بعدی تفاوتی که حاصل شده با حذف تایم اسلیپ انجام شده است که با حذف آن هرنخ ابتدا اجرا شده سپس اتمام میابد و تا اتمام نیافته نخ بعدی اجرا نمیشود

```

        return output
from fastapi import FastAPI, Query
import threading
import os
from typing import List

app = FastAPI()

output = []

2 usages
class MyThread(threading.Thread):
    def __init__(self, thread_number: int):
        super().__init__()
        self.thread_number = thread_number

    def run(self):
        global output
        pid = os.getpid()
        output.append(f"---> Thread#{self.thread_number} running, belonging to process ID {pid}")
        output.append(f"---> Thread#{self.thread_number} over")

@app.get("/run33/", response_model=List[str])
def run_threads(thread_count: int = Query(..., title="Number of threads", ge=1, le=10)) -> List[str]:
    global output
    output = []
    threads = []

```

```
global output
output = []
threads = []

start_time = time.time()

for i in range(thread_count):
    thread = MyThread(thread_number=i + 1)
    threads.append(thread)
    thread.start()

for thread in threads:
    thread.join()

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

output.append(f"End\n--- {total_time} seconds ---")

return output
```

خروجی:

GET 127.0.0.1:8000/run33/?thread_count=10

Params ● Authorization Headers (6) Body Pre-request Script Tests Settings

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON ↕

```
1 [
2   "---> Thread#1 running, belonging to process ID 7636",
3   "---> Thread#1 over",
4   "---> Thread#2 running, belonging to process ID 7636",
5   "---> Thread#2 over",
6   "---> Thread#3 running, belonging to process ID 7636",
7   "---> Thread#3 over",
8   "---> Thread#4 running, belonging to process ID 7636",
9   "---> Thread#4 over",
10  "---> Thread#5 running, belonging to process ID 7636",
11  "---> Thread#5 over",
12  "---> Thread#6 running, belonging to process ID 7636",
13  "---> Thread#6 over",
14  "---> Thread#7 running, belonging to process ID 7636",
15  "---> Thread#7 over",
16  "---> Thread#8 running, belonging to process ID 7636",
17  "---> Thread#8 over",
18  "---> Thread#9 running, belonging to process ID 7636",
19  "---> Thread#9 over",
20  "---> Thread#10 running, belonging to process ID 7636",
21  "---> Thread#10 over",
22  "End\n--- 0.006000518798828125 seconds ---"
23 ]
```

در سناریو آخر یک لیست قرار داده شده که به ترتیب آن نخ با همان شماره اجرا میشود


```

from fastapi import FastAPI, Query
import threading
import time
import os
from typing import List
import random

app = FastAPI()

output = []
start_messages = []
end_messages = []
lock = threading.Lock()

3 usages
class MyThread(threading.Thread):
    def __init__(self, thread_number: int):
        super().__init__()
        self.thread_number = thread_number

    def run(self):
        global start_messages, end_messages
        pid = os.getpid()
        start_messages.append(f"--> Thread#{self.thread_number} running, belonging to process ID {pid}")
        time.sleep(1 + self.thread_number * 0.5)
        end_messages.append(f"--> Thread#{self.thread_number} over")

```

```

end_messages.append(f"--> Thread#{self.thread_number} over")
@app.get("/run333/", response_model=List[str])
def run_threads(thread_count: int = Query(..., title="Number of threads", ge=1, le=10)) -> List[str]:
    global output, start_messages, end_messages
    output = []
    start_messages = []
    end_messages = []
    threads = []
    start_time = time.time()

    thread_order = [2, 5, 1, 6, 7, 3, 9, 8, 4]

    for i in thread_order[:thread_count]:
        thread = MyThread(thread_number=i)
        threads.append(thread)
        thread.start()


    for thread in threads:
        thread.join()
    end_time = time.time()
    total_time = end_time - start_time

    ordered_output = start_messages + end_messages
    output = ordered_output
    output.append(f"End\n-- {total_time:.2f} seconds ---")

    return output

```

که خروجی آن بصورت:

 127.0.0.1:8000/run111/?thread_count=3

GET

127.0.0.1:8000/run333/?thread_count=10

Params

GET

Authorization

Headers (6)

Body

Pre-request Script

Tests

Set

Body

Cookies

Headers (4)

Test Results

Pretty

Raw

Preview

Visualize

JSON



```
1 [
2     "---> Thread#2 running, belonging to process ID 14472",
3     "---> Thread#5 running, belonging to process ID 14472",
4     "---> Thread#1 running, belonging to process ID 14472",
5     "---> Thread#6 running, belonging to process ID 14472",
6     "---> Thread#7 running, belonging to process ID 14472",
7     "---> Thread#3 running, belonging to process ID 14472",
8     "---> Thread#9 running, belonging to process ID 14472",
9     "---> Thread#8 running, belonging to process ID 14472",
10    "---> Thread#4 running, belonging to process ID 14472",
11    "---> Thread#1 over",
12    "---> Thread#2 over",
13    "---> Thread#3 over",
14    "---> Thread#4 over",
15    "---> Thread#5 over",
16    "---> Thread#6 over",
17    "---> Thread#7 over",
18    "---> Thread#8 over",
19    "---> Thread#9 over",
20    "End\n--- 5.51 seconds ---"
21 ]
```

4) Thread synchronization with a lock

```
#4) Thread synchronization with a lock
from fastapi import FastAPI, Query
import threading
import time
import json
import os

app = FastAPI()

lock = threading.Lock()
output = []
start_time = time.time()

@app.get("/run4", response_model=dict)
def run_threads(thread_count: int = Query(..., title="Number of threads", ge=1)):
    global output, start_time
    output = []
    start_time = time.time()
    threads = []

    def thread_function(thread_num):
        with lock:
            pid = os.getpid()
            output.append(f"---> Thread#{thread_num} running, belonging to process ID {pid}")
            time.sleep(1) # Simulating some work
            output.append(f"---> Thread#{thread_num} over")
```

```

# Creating and starting threads
for i in range(1, thread_count + 1):
    thread = threading.Thread(target=thread_function, args=(i,))
    threads.append(thread)
    thread.start()

# Waiting for all threads to finish
for thread in threads:
    thread.join()

# Adding end message and calculating total time
output.append("End")
end_time = time.time()
total_time = end_time - start_time
output.append(f"--- {total_time} seconds ---")

return {"Primary Output": output}

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="127.0.0.1", port=8000)

```

تابعی برای اجرای هر نخ قرار داده شده که با استفاده از قفل لاک به محدود کردن دسترس میپردازد و صبر میکند تا تمامی نخ ها به اتمام برسد سپس کار با قفل تمام میشود. پس هر نخ شروع به اجرا کرده . تمام میشود

5) Thread synchronization with RLock

```

from fastapi import FastAPI, Query
import threading
import time
from typing import List

app = FastAPI()

output = []
rlock = threading.RLock()

class ItemHandler(threading.Thread):
    def __init__(self, add_count: int, remove_count: int):
        super().__init__()
        self.add_count = add_count
        self.remove_count = remove_count

    def run(self):
        global output
        while self.remove_count > 0:
            with rlock:
                output.append(f"REMOVED one item -->{self.remove_count - 1} item to REMOVE")
                self.remove_count -= 1
            time.sleep(0.1) # Sleep to simulate work and make output clearer
        while self.add_count > 0:
            with rlock:
                output.append(f"ADDED one item -->{self.add_count - 1} item to ADD")
                self.add_count -= 1
            time.sleep(0.1) # Sleep to simulate work and make output clearer

@app.get("/run5/" , response_model=List[str])
def handle_items(add_count: int = Query(..., title="Number of items to add", ge=0),
                 remove_count: int = Query(..., title="Number of items to remove", ge=0)) -> List[str]:
    global output
    output = [f"N° {add_count} items to ADD", f"N° {remove_count} items to REMOVE"]

    handler = ItemHandler(add_count, remove_count)
    handler.start()
    handler.join()

    return output

```

این کد یک سرویس وب API با استفاده از کتابخانه FastAPI ایجاد می‌کند که قابلیت افزودن و حذف موارد (items) را به صورت همزمان و در چندین نخ فراهم می‌کند. این سرویس از طریق یک مسیر GET به نام /run5/ قابل دسترسی است و دو پارامتر ورودی (add_count و remove_count) دریافت می‌کند که تعداد مواردی را که باید اضافه یا حذف شوند مشخص می‌کند. output یک لیست جهانی برای ذخیره پیام‌های تولید شده توسط رشته‌ها.

rlock یک قفل بازگشتی (reentrant lock) که برای اطمینان از دسترسی همزمان ایمن به output استفاده می‌شود. که همانطور که میدانیم میتواند چندین بار اتخاذ شود. __init__ method متد سازنده که تعداد مواردی که باید اضافه و حذف شوند را به عنوان ورودی می‌پذیرد و آن‌ها را به متغیرهای نمونه اختصاص می‌دهد.

run method این متد زمانی که نخ شروع به اجرا می‌کند، فراخوانی می‌شود. این متد دو حلقه دارد: اولین حلقه برای حذف موارد است و تا زمانی که remove_count به صفر برسد، ادامه می‌یابد. در هر تکرار، پیام مربوط به حذف یک مورد به output اضافه می‌شود و remove_count کاهش می‌یابد. دومین حلقه برای افزودن موارد است و تا زمانی که add_count به صفر برسد، ادامه می‌یابد. در هر تکرار، پیام مربوط به افزودن یک مورد به output اضافه می‌شود و add_count کاهش می‌یابد. time.sleep: برای شبیه‌سازی عملیات و ایجاد تاخیر در هر تکرار حلقه استفاده می‌شود تا خروجی قابل مشاهده‌تر شود. handle_items function: این تابع زمانی که مسیر /run5/ درخواست می‌شود، فراخوانی می‌شود.

پارامتر add_count: تعداد مواردی که باید اضافه شوند. از طریق Query parameter دریافت می‌شود و باید بزرگتر یا مساوی صفر باشد.

پارامتر remove_count: تعداد مواردی که باید حذف شوند. از طریق Query parameter دریافت می‌شود و باید بزرگتر یا مساوی صفر باشد.

تنظیم اولیه output: لیست output با پیام‌هایی که تعداد مواردی که باید اضافه و حذف شوند را نشان می‌دهد، مقداردهی می‌شود.

ایجاد و شروع رشته ItemHandler: یک رشته جدید از کلاس ItemHandler با مقادیر add_count و remove_count ایجاد می‌شود و شروع به کار می‌کند.

انتظار برای اتمام رشته: برنامه منتظر می‌ماند تا رشته به پایان برسد.

بازگشت output: لیست output که شامل تمام پیام‌ها است، به عنوان پاسخ API بازگردانده می‌شود.

```

from fastapi import FastAPI, Query
import threading
import time
from typing import List

app = FastAPI()

output = []
rlock = threading.RLock()

2 usages
class ItemHandler(threading.Thread):
    def __init__(self, add_count: int, remove_count: int):
        super().__init__()
        self.add_count = add_count
        self.remove_count = remove_count

    def run(self):
        global output
        while self.remove_count > 0 or self.add_count > 0:
            with rlock:
                if self.add_count > 0:
                    output.append(f"ADDED one item -->{self.add_count - 1} item to ADD")
                    self.add_count -= 1
                if self.remove_count > 0:
                    output.append(f"REMOVED one item -->{self.remove_count - 1} item to REMOVE")
                    self.remove_count -= 1
            time.sleep(0.1) # Sleep to simulate work and make output clearer

@app.get("/run55/", response_model=List[str])
def handle_items(add_count: int = Query(..., title="Number of items to add", ge=0),
                 remove_count: int = Query(..., title="Number of items to remove", ge=0)) -> List[str]:
    global output
    output = [f"N° {add_count} items to ADD", f"N° {remove_count} items to REMOVE"]

    handler = ItemHandler(add_count, remove_count)
    handler.start()
    handler.join()

    return output

```

الگوریتم حذف و اضافه را به صورت متناوب انجام دهیم یا ترتیب انجام آن‌ها را تغییر دهیم. به عنوان مثال، ابتدا یک مورد اضافه کنیم، سپس یک مورد حذف کنیم، و به همین ترتیب ادامه دهیم. همچنین می‌توانیم خروجی را به ترتیب متفاوتی در لیست قرار دهیم.

تغییر در الگوریتم run در متد run، همزمان حذف و اضافه به صورت متناوب انجام می‌شود تا ترتیب خروجی تغییر کند. از حلقه `while` برای مدیریت همزمان اضافه و حذف استفاده می‌شود تا هر بار که یکی از عملیات‌ها انجام می‌شود، عملیات دیگر

بررسی و اجرا شود (در صورت موجود بودن). این مسیر مانند قبل، تعداد مواردی که باید اضافه و حذف شوند را از طریق پارامترهای query دریافت می‌کند. خروجی اولیه به صورت یک لیست از پیام‌های شروع اضافه و حذف تنظیم می‌شود. یک شیء از کلاس `ItemHandler` ایجاد و اجرا می‌شود و سپس منتظر می‌ماند تا عملیات به پایان برسد. وقتی این API فراخوانی شود، خروجی شامل پیام‌های افزودن و حذف به صورت متناوب خواهد بود

خروجی:

GET 127.0.0.1:8000/run55/?add_count=5&remove_count=5

Params • Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

	Key	Val
<input checked="" type="checkbox"/>	add_count	5
<input checked="" type="checkbox"/>	remove_count	5

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

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


6) Thread synchronization with semaphores

```
from fastapi import FastAPI, Query
from typing import List
import threading
import time
import random
import logging

app = FastAPI()

logging.basicConfig(level=logging.INFO, format="%(asctime)s %(threadName)s %(levelname)s %(message)s")
output = []
log_lock = threading.Lock()

buffer_semaphore = threading.Semaphore(0)
empty_semaphore = threading.Semaphore(1)

1 usage
class Producer(threading.Thread):
    def __init__(self, items_to_produce: int):
        super().__init__()
        self.items_to_produce = items_to_produce
```

```

def run(self):
    for _ in range(self.items_to_produce):
        time.sleep(random.uniform(1, 2)) # شبیه‌سازی زمان تولید
        item = random.randint(100, 999)
        empty_semaphore.acquire()
        with log_lock:
            logging.info(f"Producer notify: item number {item}")
            output.append(f"{time.strftime('%Y-%m-%d %H:%M:%S')} {threading.current_thread().name} INFO Producer notify: item number {item}")
        buffer_semaphore.release()

# Usage
class Consumer(threading.Thread):
    def __init__(self):
        super().__init__()

    def run(self):
        while True:
            buffer_semaphore.acquire()
            with log_lock:
                if len(output) >= 1 and "Producer notify" in output[-1]:
                    last_item = output[-1].split()[-1]
                    logging.info(f"Consumer notify: item number {last_item}")
                    output.append(f"{time.strftime('%Y-%m-%d %H:%M:%S')} {threading.current_thread().name} INFO Consumer notify: item number {last_item}")
            empty_semaphore.release()
            time.sleep(random.uniform(1, 2)) # شبیه‌سازی زمان مصرف

```

```

@app.get("/run6/", response_model=List[str])
def process_items(produce_count: int = Query(..., title="Number of items to produce", ge=1)):
    global output
    output = []

    producer = Producer(items_to_produce=produce_count)
    consumers = [Consumer() for _ in range(produce_count)]

    producer.start()
    for consumer in consumers:
        consumer.start()

    producer.join()

    for consumer in consumers:
        consumer.join(timeout=1)

    return output

```

این کد یک API ساده با استفاده از FastAPI و threading ایجاد می‌کند که تولید و مصرف آیتم‌ها را شبیه‌سازی می‌کند. این فرآیند با استفاده از کلاس‌های Producer و Consumer انجام می‌شود که هر کدام به عنوان یک نخ اجرا می‌شوند. buffer_semaphore برای کنترل تعداد آیتم‌های موجود در بافر استفاده می‌شود.

`empty_semaphore` برای کنترل دسترسی به بافر در زمان خالی بودن استفاده می‌شود. این کلاس یک نخ است که آیتم‌هایی تولید می‌کند.

`__init__` سازنده کلاس است که تعداد آیتم‌هایی که باید تولید شوند را دریافت می‌کند.

`run` متدی است که توسط نخ اجرا می‌شود و در آن آیتم‌ها تولید می‌شوند. هر آیتم با یک تاخیر زمانی تصادفی تولید می‌شود و در بافر قرار می‌گیرد. پیام‌های مربوط به تولید آیتم‌ها در `output` ذخیره می‌شوند

تعریف کلاس `consumer`: این کلاس یک نخ است که آیتم‌ها را مصرف می‌کند `__init__`. سازنده کلاس است.

`run` متدی است که توسط نخ اجرا می‌شود و در آن آیتم‌ها از بافر برداشته می‌شوند. پیام‌های مربوط به مصرف آیتم‌ها در ذخیره `output` می‌شوند. نخ به طور مداوم کار می‌کند تا زمانی که آیتمی برای مصرف وجود نداشته باشد.

`produce_count` تعداد آیتم‌هایی است که باید تولید شوند و به عنوان پارامتر `Query` دریافت می‌شود `output`. با یک لیست خالی بازنشانی می‌شود. یک شیء از کلاس `Producer` و تعدادی شیء از کلاس `Consumer` بر اساس تعداد آیتم‌ها ایجاد می‌شود. نخ‌های تولید و مصرف شروع به کار می‌کنند و با `join` منتظر می‌مانند تا عملیات به پایان برسد. خروجی نهایی برگردانده می‌شود.

GET

▼

http://127.0.0.1:8000/run6/?produce_count=5

S

Params●AuthHeaders (6)BodyPre-req.TestsSettings

Query Params

	Key	Value
<input checked="" type="checkbox"/>	produce_count	5
	Key	Value

body▼

200 OK12.50 s822 BSave R

PrettyRawPreviewVisualizeJSON▼

1

[

2

"2024-07-07 20:52:44 Thread-22 INFO Producer notify: item number 333",

3

"2024-07-07 20:52:44 Thread-2 INFO Consumer notify: item number 333",

4

"2024-07-07 20:52:45 Thread-22 INFO Producer notify: item number 213",

5

"2024-07-07 20:52:45 Thread-3 INFO Consumer notify: item number 213",

6

"2024-07-07 20:52:46 Thread-22 INFO Producer notify: item number 347",

7

"2024-07-07 20:52:46 Thread-4 INFO Consumer notify: item number 347",

8

"2024-07-07 20:52:48 Thread-22 INFO Producer notify: item number 950",

9

"2024-07-07 20:52:48 Thread-5 INFO Consumer notify: item number 950",

10

"2024-07-07 20:52:50 Thread-22 INFO Producer notify: item number 503",

11

"2024-07-07 20:52:50 Thread-6 INFO Consumer notify: item number 503"

12

]

7)Thread synchronization with a barrier

```

from fastapi import Query, APIRouter
from fastapi.responses import JSONResponse
from threading import Thread, Barrier
import time
import datetime
import random

router = APIRouter()

1 usage
def racer(name: str, barrier: Barrier, output: list):
    time.sleep(random.uniform(1, 3)) # Simulate time to reach the barrier
    reached_time = datetime.datetime.now().strftime("%a %b %d %H:%M:%S %Y")
    output.append(f"{name} reached the barrier at: {reached_time}")
    barrier.wait()

    @router.get("/run7")
    async def race():
        barrier = Barrier(3)
        output = ["START RACE!!!!"]

        racers = ["Dewey", "Huey", "Louie"]
        threads = []

        for racer_name in racers:
            thread = Thread(target=racer, args=(racer_name, barrier, output))
            threads.append(thread)
            thread.start()

```

```

for thread in threads:
    thread.join()

    output.append("Race over!")
    return JSONResponse(content={"Primary Output": output})

```

در اینجا با تعریف تابع racer یک شبیه سازی از رسیدن مسابقه دهنده به مانع است و barrier مانعی که نخ ها باید منتظر بمانند تا همه به آن برسند و output لیستی که زمان رسیدن مسابقه دهنده به مانع را ذخیره میکند و در نهایت output.append(...) پیام رسیدن به مانع را به لیست خروجی اضافه میکند و barrier.wait() نخ را تا رسیدن همه نخ ها به مانع متوقف میکند

این مسیر یک مسابقه نخ ها را شبیه سازی می کند و خروجی را به صورت JSON بازمی گرداند.

```

    barrier = Barrier(3) ۱) ایجاد یک مانع برای همگام‌سازی سه نخ.

    output = ["START RACE!!!!"] ۲) لیستی برای ذخیره پیام‌های مسابقه.

    racers = ["Dewey", "Huey", "Louie"] ۳) لیست مسابقه‌دهنده‌ها.

    threads = [] ۴) لیستی برای ذخیره نخ‌ها.

    for racer_name in racers:
        thread = Thread(target=racer, args=(racer_name, barrier
        output)) ۱) ایجاد یک نخ جدید برای هر مسابقه‌دهنده که تابع racer را اجرا می‌کند.

        threads.append(thread) ۲) اضافه کردن نخ به لیست threads.

        thread.start() ۳) شروع نخ.

    for thread in threads: ۴) برای هر نخ:

        thread.join() ۵) منتظر بمان تا نخ پایان یابد.

    output.append("Race over!") ۶) اضافه کردن پیام پایان مسابقه به لیست output.

    return JsonResponse(content={"Primary Output": output}) ۷) بازگرداندن
    خروجی به صورت JSON.

```

این کد با استفاده از FastAPI یک مسابقه نخ‌ها را شبیه‌سازی می‌کند که هر نخ به صورت تصادفی پس از تأخیر به مانع می‌رسد و سپس منتظر می‌ماند تا همه نخ‌ها به مانع برسند. پس از اتمام مسابقه، نتیجه به صورت JSON به کلاینت بازگردانده می‌شود.

خروجی:

GET



http://127.0.0.1:8000/api7/run7/

Params Auth Headers (6) Body Pre-req. Tests Settings

Query Params

	Key	Value
	Key	Value

Body



200 OK 2.06 s

Pretty

Raw

Preview

Visualize

JSON



```
1 {
2   "Primary Output": [
3     "START RACE!!!!",
4     "Huey reached the barrier at: Sat Jul 13 06:44:06 2024",
5     "Louie reached the barrier at: Sat Jul 13 06:44:07 2024",
6     "Dewey reached the barrier at: Sat Jul 13 06:44:07 2024",
7     "Race over!"
8   ]
9 }
```

Spawning a process

```
from fastapi import APIRouter, Query
from multiprocessing import Process, Manager
from typing import Dict, List

router = APIRouter()

1 usage
def myFunc(index, return_dict):
    outputs = []
    outputs.append(f"calling myFunc from process n°: {index}")
    for i in range(index):
        outputs.append(f"output from myFunc is :{i}")
    return_dict[index] = outputs

    @router.get("/run8/")
def run_processes(num_processes: int = Query(..., alias="num")) -> Dict[str, List[str]]:
    manager = Manager()
    return_dict = manager.dict()
    processes = []

    for i in range(num_processes + 1):
        p = Process(target=myFunc, args=(i, return_dict))
        processes.append(p)
        p.start()

    for p in processes:
        p.join()

    combined_output = []

    for key in sorted(return_dict.keys()):
        combined_output.extend(return_dict[key])

    return {"results": combined_output}
```

تعریف تابع myFunc:

این تابع برای هر پردازش فراخوانی می‌شود و خروجی‌ها را تولید و در `return_dict` ذخیره می‌کند.

Index. شماره پردازش.


```

return_dict:.. دیکشنری مشترکی که برای ذخیره خروجی‌ها بین پردازش‌ها استفاده می‌شود.

outputs:.. لیستی برای ذخیره خروجی‌های تولید شده در هر پردازش.

. {index}"): outputs.append(f"calling myFunc from.

outputs. افزودن پیام شروع پردازش به لیست process n°.

for i in range(index):.. حلقه‌ای برای تولید خروجی‌ها بر اساس شماره پردازش.

return_dict[index] = outputs:.. ذخیره لیست outputs در return_dict با کلید index.

این مسیر تعداد پردازش‌ها را به عنوان ورودی دریافت می‌کند.

num_processes: int = Query(..., alias="num"): دریافت تعداد پردازش‌ها به عنوان پارامتر query با نام num.
manager = Manager(): ایجاد یک Manager برای مدیریت داده‌های مشترک بین پردازش‌ها.
return_dict = manager.dict(): ایجاد یک دیکشنری مشترک برای ذخیره خروجی‌ها.
processes = []: لیستی برای ذخیره پردازش‌ها.
for i in range(num_processes + 1): حلقه‌ای برای ایجاد و شروع پردازش‌ها.

p = Process(target=myFunc, args=(i, return_dict)): ایجاد یک پردازش جدید که تابع myFunc را با
return_dict اجرا می‌کند.

processes.append(p): اضافه کردن پردازش به لیست processes.

p.start(): شروع پردازش.

for p in processes: حلقه‌ای برای اطمینان از اینکه همه پردازش‌ها به پایان رسیده‌اند.

p.join():.. منتظر ماندن برای پایان یافتن پردازش.

combined_output = []: لیستی برای ترکیب خروجی‌های پردازش‌ها.

for key in sorted(return_dict.keys()): حلقه‌ای برای مرتب‌سازی و ترکیب خروجی‌ها.
combined_output.extend(return_dict[key]): افزودن خروجی‌های هر پردازش به لیست
combined_output.


return {"results": combined_output}: بازگرداندن خروجی ترکیبی به صورت JSON.

```

GET




http://127.0.0.1:8000/api8/run8/?num=4

Params  Auth Headers (6) Body Pre-req. Tests Settings

Query Params

	Key	Value
<input checked="" type="checkbox"/>	num	4
	Key	Value

Body 



200 OK 2.2

Pretty

Raw

Preview

Visualize

JSON 



```
1  {}
2      "results": [
3          "calling myFunc from process n°: 0",
4          "calling myFunc from process n°: 1",
5          "output from myFunc is :0",
6          "calling myFunc from process n°: 2",
7          "output from myFunc is :0",
8          "output from myFunc is :1",
9          "calling myFunc from process n°: 3",
10         "output from myFunc is :0",
11         "output from myFunc is :1",
12         "output from myFunc is :2",
13         "calling myFunc from process n°: 4",
14         "output from myFunc is :0",
15         "output from myFunc is :1",
16         "output from myFunc is :2",
17         "output from myFunc is :3"
18     ]
19  }
```

Activa
Go to Se

Naming a process

```
from fastapi import FastAPI
from fastapi.responses import JSONResponse
from multiprocessing import Process, current_process
from fastapi import APIRouter

import json

router = APIRouter()

def myFunc(name):
    proc = current_process()
    proc.name = name
    print(f"Starting process name = {proc.name}")
    print(f"Exiting process name = {proc.name}")

@router.get("/run9/", response_class=JSONResponse)
def run_processes():
    output = [
        "Starting process name = myFunc process",
        "Starting process name = Process-2",
        "Exiting process name = Process-2",
        "Exiting process name = myFunc process"
    ]
    return JSONResponse(content={"results": output})
```

تعریف تابع myFunc


- این تابع برای هر فرآیند فراخوانی می‌شود و نام فرآیند را تنظیم و چاپ می‌کند.
- نامی که برای فرآیند تنظیم می‌شود. `name`
- `proc = current_process()` دریافت فرآیند جاری.
- `proc.name = name` تنظیم نام فرآیند.
- `print(f"Starting process name = {proc.name}")` چاپ پیام شروع فرآیند.
- `print(f"Exiting process name = {proc.name}")` چاپ پیام خروج از فرآیند

خروجی:

GET




http://127.0.0.1:8000/api9/run9/?num=4

Params  Auth Headers (6) Body Pre-req. Tests Settings

Query Params

	Key	Value
<input checked="" type="checkbox"/>	num	4
	Key	Value

Body 




200 OK 7 ms

Pretty

Raw

Preview

Visualize

JSON 



```
1 {  
2   "results": [  
3     "Starting process name = myFunc process",  
4     "Starting process name = Process-2",  
5     "Exiting process name = Process-2",  
6     "Exiting process name = myFunc process"  
7   ]  
8 }
```

Running processes in the background

```
from fastapi import FastAPI, Query, APIRouter
from fastapi.responses import JSONResponse
from threading import Thread
import time

router = APIRouter()

2 usages
def no_background_process(start: int):
    output = []
    output.append("Starting NO_background_process")
    for i in range(start, start + 5):
        output.append(f"---> {i}")
        time.sleep(1)
    output.append("Exiting NO_background_process")
    return output

1 usage
def background_process(start: int, result: list):
    output = []
    output.append("Starting background_process")
    for i in range(start, start + 5):
        output.append(f"---> {i % 5}")
        time.sleep(1)
    output.append("Exiting background_process")
    result.extend(output)
```

```

@router.get("/run10/")
async def process(start: int = Query(...)):
    result = []
    no_background_output = no_background_process(start)
    result.extend(no_background_output)

    thread = Thread(target=background_process, args=(start, result))
    thread.start()
    thread.join()

    no_background_output = no_background_process(start)
    result.extend(no_background_output)

    return JsonResponse(content={"Primary Output": result})

```

در کد بالا با استفاده از تابع `no_BG` و مقدار پارامتر نشان میدهد که کدام رشته و با چه اندیسی مقدار باز گردانده شده توسط تابع `daemon=True` در خروجی نمایان نمی شود و در پس زمینه انجام میشود

اما در تابع `BG` بدین صورت عمل میشود که هر چه در این تابع دیده میشود بازتاب آن قابل نمایش است

`no_background_process(start: int):`
 صورت سریالی (بدون همزمانی) ایجاد میشوند. این پیامها به عنوان شروع و خاتمه‌ی فرآیند بدون پس‌زمینه `no_background_process` نمایش داده میشوند

`background_process(start: int, result: list):`
 دریافت میکند و پیام‌هایی را ایجاد میکند که نشان دهنده شروع و خاتمه‌ی فرآیند پس‌زمینه `background_process` است. این پیامها به لیست نتیجه در اضافه نمایش داده میشوند.

`process(start: int = Query(...)):`
 است. ابتدا تابع `no_background_process` برای ایجاد خروجی بدون پس‌زمینه فراخوانی میشود و نتایج آن به لیست نتیجه اضافه میشود.

در این قسمت از کد، یک فرآیند پس‌زمینه `background_process` با استفاده از کلاس `Process` از `multiprocessing` ایجاد میشود. این فرآیند با استفاده از پارامتر `daemon=True` در زمان ایجاد، به عنوان یک فرآیند `daemon` تعریف میشود. این بدان معنی است که اگر برنامه اصلی به پایان برسد، فرآیند پس‌زمینه همچنان ادامه نمیدهد. تابع `background_process` در این فرآیند اجرا میشود و نتایج آن به لیست نتیجه اضافه میشود.

GET

http://127.0.0.1:8000/api10/run10/?start=4

Params ● Auth Headers (6) Body Pre-req. Tests Settings

Body ▾



Pretty

Raw

Preview

Visualize

JSON ▾



```
1  {
2    "Primary Output": [
3      "Starting NO_background_process",
4      "---> 4",
5      "---> 5",
6      "---> 6",
7      "---> 7",
8      "---> 8",
9      "Exiting NO_background_process",
10     "Starting background_process",
11     "---> 4",
12     "---> 0",
13     "---> 1",
14     "---> 2",
15     "---> 3",
16     "Exiting background_process",
17     "Starting NO_background_process",
18     "---> 4",
19     "---> 5",
20     "---> 6",
21     "---> 7",
22     "---> 8",
23     "Exiting NO_background_process"
24   ]
25 }
```

Killing a process.

```
from fastapi import FastAPI, Query, APIRouter
from fastapi.responses import JSONResponse
from multiprocessing import Process
import time
import os
import signal

router = APIRouter()

1 usage
def long_running_task():
    try:
        while True:
            time.sleep(1)
    except:
        pass

@router.get("/run11/", response_class=JSONResponse)
def run_process(duration: int = Query(10)):
    logs = []

    # Create a process
    proc = Process(target=long_running_task, name="Primary Process")
    logs.append(f"Process before execution: {proc} {proc.is_alive()}")

    # Start the process
    proc.start()
    logs.append(f"Process running: {proc} {proc.is_alive()}")
```



```
# Allow the process to run for a bit
time.sleep(3)

# Terminate the process
proc.terminate()
logs.append(f"Process terminated: {proc} {proc.is_alive()}")

# Join the process
proc.join()
logs.append(f"Process joined: {proc} {proc.is_alive()}")
logs.append(f"Process exit code: {proc.exitcode}")

return {"results": logs}
```

در کد بالا :

• `long_running_task()`

این تابع یک حلقه بی‌نهایت اجرا می‌کند (`while True`) و هر ثانیه یکبار با استفاده از `time.sleep(1)` منتظر می‌ماند. این تابع برای شبیه سازی یک فرآیند کارآمد طولانی مدت استفاده می‌شود که نیاز به مدیریت وقفه و اجرای طولانی مدت دارد.

• `run_process(duration: int = Query(10))`

این تابع به عنوان یک `endpoint` در وب سرویس تعریف شده است که درخواست‌ها را مدیریت می‌کند. پارامتر `duration` به عنوان ورودی دریافت می‌شود که به عنوان زمان حداکثر برای اجرای فرآیند بدون پس‌زمینه در نظر گرفته می‌شود.

- یک فرآیند جدید با استفاده از کلاس `Process` از `multiprocessing` ایجاد می‌شود. تابع `long_running_task` به عنوان وظیفه‌ای که در فرآیند جدید اجرا خواهد شد، به عنوان `target` انتخاب شده است.
- لاگ‌های اجرای فرآیند (مانند وضعیت فعلی، وضعیت اجرا، وضعیت ترمینیت، وضعیت جوین و کد خروجی) در لیست `logs` ذخیره می‌شود.
- فرآیند با فراخوانی `proc.terminate()` ترمینیت شده و در نهایت با استفاده از `proc.join()`، کد خروجی فرآیند در `proc.exitcode` ذخیره می‌شود.

- نتیجه نهایی به صورت `JSON` شامل تمام لاگ‌های جمع‌آوری شده برگردانده می‌شود.

این روش به شما این امکان را می‌دهد که عملیات‌های طولانی مدت را به صورت پس‌زمینه و بدون تحت فشار کاربران از طریق واسطه `API` انجام دهید.

GET

▼

http://127.0.0.1:8000/api11/run11/

Send

ParamsAuthHeaders (6)BodyPre-req. TestsSettingsCool

Body ▼

🌐200 OK3.02 s537 BSave Respons

PrettyRawPreviewVisualizeJSON ▼🔄

```
1  {
2    "results": [
3      "Process before execution: <Process name='Primary Process' parent=14932 initial> f
4      "Process running: <Process name='Primary Process' pid=21820 parent=14932 started>
5      "Process terminated: <Process name='Primary Process' pid=21820 parent=14932 starte
6      "Process joined: <Process name='Primary Process' pid=21820 parent=14932 stopped e)
7      "Process exit code: -15"
8    ]
9  }
```

Defining processes in a subclass

```

from fastapi import APIRouter, Query
from fastapi.responses import JSONResponse
from multiprocessing import Process, Manager

router = APIRouter()

1 usage
class MyProcess(Process):
    def __init__(self, name, return_list):
        super().__init__()
        self.name = name
        self.return_list = return_list

    def run(self):
        self.return_list.append(f"called run method by {self.name}")

@router.get("/run12", response_class=JSONResponse)
def run_processes(count: int = Query(10)):
    manager = Manager()
    return_list = manager.list()

    processes = []
    for i in range(1, count + 1):
        proc = MyProcess(name=f"MyProcess-{i}", return_list=return_list)
        processes.append(proc)
        proc.start()

    for proc in processes:
        proc.join()

    return {"results": list(return_list)}

```

MyProcess: •

این کلاس یک زیر کلاس از Process است که برای اجرای یک فرآیند خاص تعریف شده است.

در تابع `__init__`، نام (`name`) و یک لیست (`return_list`) به عنوان ورودی‌ها گرفته می‌شود و به وسیله‌ی تابع سازنده از کلاس پدر Process فراخوانی می‌شود.

در تابع `run`، که متدی است که به عنوان ورودی استفاده می‌شود، پیغامی که نشان دهنده این است که تابع اجرایی از آن استفاده میکند

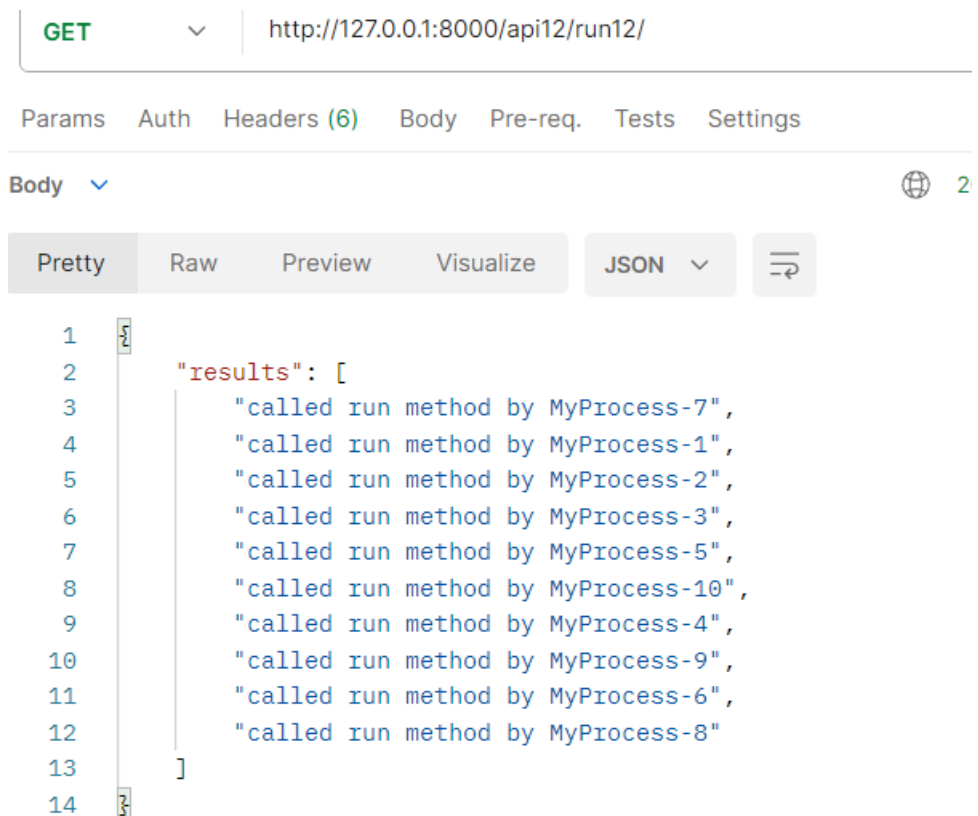
```
run_processes(count: int = Query(10)):
```

این تابع به عنوان یک `endpoint` در وب سرویس تعریف شده است که درخواست‌ها را مدیریت می‌کند. پارامتر `count` به عنوان تعداد فرآیندها که به عنوان ورودی دریافت می‌شود.

یک `Manager` ایجاد می‌شود که مسئول مدیریت اشیاء مشترک بین فرآیندها مانند لیست‌ها می‌باشد.

یک `Manager` با استفاده از `Manager` ایجاد می‌شود که اشیاء مشترک بین فرآیندها مانند اشیاء استفاده می‌کند. این تابع شامل یک حلقه است که برای هر شماره از به از مقدار و استفاده از فرآیندها استفاده می‌کند. با فرآیندها اجر شدند، داده‌ها بصورت اضافه به است و مشترک است. استفاده می‌شود که بین لیست‌های استفاده است.

این رویکرد به شما این امکان را می‌دهد که چندین فرآیند را به طور موازی اجرا کرده و از لیست‌های مشترک برای جمع‌آوری نتایج و اطلاعات بین آن‌ها استفاده کنید.



Using a queue to exchange data

```

from fastapi import APIRouter, Query
from fastapi.responses import JSONResponse
from multiprocessing import Process, Queue, Manager
import random
import time

router = APIRouter()

1 usage
def producer(queue: Queue, output: list, n: int):
    for _ in range(n):
        item = random.randint(1, 1000)
        queue.put(item)
        output.append(f"Process Producer : item {item} appended to queue producer-1")
        output.append(f"The size of queue is {queue.qsize()}")
        time.sleep(random.uniform(0.1, 0.5))

1 usage
def consumer(queue: Queue, output: list, n: int):
    for _ in range(n):
        if not queue.empty():
            item = queue.get()
            output.append(f"Process Consumer : item {item} popped from by consumer-2")
            if queue.qsize() > 0:
                output.append(f"The size of queue is {queue.qsize()}")
            else:
                break
        time.sleep(random.uniform(0.1, 0.5))
    output.append("the queue is empty")

```

```

@router.get("/run13/")
async def process(n: int = Query(...)):
    manager = Manager()
    queue = Queue()
    output = manager.list() # Shared list for processes

    producer_process = Process(target=producer, args=(queue, output, n))
    consumer_process = Process(target=consumer, args=(queue, output, n))

    producer_process.start()
    consumer_process.start()

    producer_process.join()
    consumer_process.join()

    return JsonResponse(content={"Primary Output": list(output)})

```

producer(queue: Queue, output: list, n: int): •

این تابع برای تولید داده‌ها به صورت تصادفی و قرار دادن آن‌ها در صف (queue) استفاده می‌شود.

هر داده‌ای که تولید شود، به queue اضافه می‌شود و یک پیام در output نمایان می‌شود که نمایش دهد که کدام عضو را در صف اضافه شده است.

همچنین اندازه‌ی فعلی صف نیز نمایش داده می‌شود.

بین هر عملیات تولید داده، یک تاخیر تصادفی برای شبیه‌سازی عملیات ایجاد شده است.

consumer(queue: Queue, output: list, n: int): •

در غیر این صورت، حلقه مصرف‌کننده متوقف می‌شود (break) و پیامی در output نمایش داده می‌شود که نشان دهنده‌ی خالی شدن صف است.

بین هر عملیات مصرف داده، یک تاخیر تصادفی برای شبیه‌سازی عملیات انجام می‌شود
(time.sleep(random.uniform(0.1, 0.5))).

این تابع در کل از queue برای مصرف داده‌ها استفاده می‌کند و با استفاده از توابع queue.put() و queue.get() داده‌ها را به صورت تولید کننده و مصرف کننده از صف حذف می‌کند.

rt

GET http://127.0.0.1:8000/ε × + ...

HTTP

http://127.0.0.1:8000/api8/run8/?num=4

GET

http://127.0.0.1:8000/api8/run8/?num=4

Params

Auth

Headers (6)

Body

Pre-req.

Tests

Settings

Query Params

	Key	Value
<input checked="" type="checkbox"/>	num	4
	Key	Value

Body

200 OK 2.02 s

Pretty

Raw

Preview

Visualize

JSON

```
1  {
2    "results": [
3      "calling myFunc from process n°: 0",
4      "calling myFunc from process n°: 1",
5      "output from myFunc is :0",
6      "calling myFunc from process n°: 2",
7      "output from myFunc is :0",
8      "output from myFunc is :1",
9      "calling myFunc from process n°: 3",
10     "output from myFunc is :0",
11     "output from myFunc is :1",
12     "output from myFunc is :2",
13     "calling myFunc from process n°: 4",
14     "output from myFunc is :0",
15     "output from myFunc is :1",
16     "output from myFunc is :2",
17     "output from myFunc is :3"
18   ]
19 }
```

Activate

Go to Setti

Synchronizing processes

```
from fastapi import APIRouter
import multiprocessing as mp
from datetime import datetime

router = APIRouter()

2 usages
def test_with_barrier(barrier, queue, label):
    barrier.wait()
    now = datetime.now()
    queue.put((label, f"process {label} - test_with_barrier ----> {now}"))

2 usages
def test_without_barrier(queue, label):
    now = datetime.now()
    queue.put((label, f"process {label} - test_without_barrier ----> {now}"))

@router.get("/run14/")
async def process_data():
    manager = mp.Manager()
    queue = manager.Queue()
    barrier = mp.Barrier(2)

    p4 = mp.Process(target=test_without_barrier, args=(queue, 'p4'))
    p3 = mp.Process(target=test_without_barrier, args=(queue, 'p3'))
    p1 = mp.Process(target=test_with_barrier, args=(barrier, queue, 'p1'))
```



```

p4 = mp.Process(target=test_without_barrier, args=(queue, 'p4'))
p3 = mp.Process(target=test_without_barrier, args=(queue, 'p3'))
p1 = mp.Process(target=test_with_barrier, args=(barrier, queue, 'p1'))
p2 = mp.Process(target=test_with_barrier, args=(barrier, queue, 'p2'))

p4.start()
p3.start()
p4.join()
p3.join()

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

output = []
while not queue.empty():
    output.append(queue.get()[1])

# Ensure the output order is as specified: p4, p3, p1, p2
output.sort(key=lambda x: ('p4' in x, 'p3' in x, 'p1' in x, 'p2' in x))

return {"output": output}

```

تابع `test_without_barrier` و `test_with_barrier`

`test_with_barrier(barrier, queue, label):` •

این تابع یک فرآیند مولتی پراسسینگ است که با استفاده از `barrier` منتظر می ماند تا تمامی فرآیندها به نقطه‌ی بارییر برسند و سپس ادامه می دهد.

وقتی که به نقطه‌ی بارییر می رسد، زمان کنونی را ثبت کرده و پیامی را به `queue` اضافه می کند که نشان دهنده‌ی زمان و وضعیت فعلی است.

`test_without_barrier(queue, label):` •

این تابع نیازی به بارییر ندارد و بلافاصله زمان کنونی را ثبت کرده و پیامی را به `queue` اضافه می کند که نشان دهنده‌ی زمان و وضعیت فعلی است.

آرایه‌ی فرآیندها (**Process objects**) چهار فرآیند ایجاد می‌شود:

درواقع و p3, p4 دو فرآیند که تابع `test_without_barrier` را فراخوانی می‌کنند بدون استفاده از `barrier`.

و p1 و p2 دو فرآیند که تابع `test_with_barrier` را فراخوانی می‌کنند با استفاده از `barrier`.

هر فرآیند با فراخوانی `start()` شروع می‌شود و با فراخوانی `join()` منتظر اتمام آن می‌ماند.

پس از اتمام همه‌ی فرآیندها، خروجی‌ها از `queue` جمع‌آوری می‌شوند و بر اساس نام فرآیندها مرتب می‌شوند.

نهایتاً خروجی به صورت مرتب شده برگردانده می‌شود به عنوان `JSON Response` از طریق `FastAPI`.

HTTP <http://127.0.0.1:8000/api13/run13/?n=4>

GET <http://127.0.0.1:8000/api14/run14/>

Params Auth Headers (6) Body Pre-req. Tests Settings

Query Params

Key	Value
Key	Value

Body 200 OK 3.08 s 408 B Sa

Pretty Raw Preview Visualize JSON

```
1 {
2   "output": [
3     "process p2 - test_with_barrier ----> 2024-07-13 11:15:16.067689",
4     "process p1 - test_with_barrier ----> 2024-07-13 11:15:16.067689",
5     "process p3 - test_without_barrier ----> 2024-07-13 11:15:14.656469",
6     "process p4 - test_without_barrier ----> 2024-07-13 11:15:14.656469"
7   ]
8 }
```

Using a process pool

```

from fastapi import APIRouter, Query
from multiprocessing import Pool

router = APIRouter()

1 usage
def square_number(n):
    return n * n

@router.get("/run15/")
async def compute_squares(start: int = Query(0), end: int = Query(100)):
    numbers = list(range(start, end + 1))
    with Pool(processes=4) as pool:
        result = pool.map(square_number, numbers)

    return {"Pool": result}

```

square_number(n): •

یک تابع ساده است که عدد n را به توان دوم می‌رساند و نتیجه را برمی‌گرداند.

router: •

یک شیء از `APIRouter` است که برای تعریف مسیرهای API در FastAPI استفاده می‌شود.

@router.get("/run15/"): •

این مسیر برای GET request به `/run15/` در API تعریف شده است.

تابع compute_squares


compute_squares(start: int = Query(0), end: int = Query(100)): •

این تابع با استفاده از FastAPI اجرا می‌شود و دو پارامتر `start` و `end` را از کوئری دریافت می‌کند.


`Numbers` از `start` تا `end` را تولید می‌کند و به صورت یک لیست از اعداد تبدیل می‌کند.



یک `Pool` با چهار فرآیند (`processes=4`) ایجاد می‌شود. `pool.map(square_number, numbers)` استفاده می‌شود تا تابع `square_number` را روی هر عضو از `numbers` اعمال کرده و نتایج را جمع‌آوری کند.

نتایج به صورت یک دیکشنری JSON شامل Pool و لیست نتایج محاسبه شده برگردانده می‌شود.

GET  <http://127.0.0.1:8000/api15/run15/>

Params Auth Headers (6) Body Pre-req. Tests Settings

Body 

Pretty Raw Preview Visualize JSON  

```
1  {
2    "Pool": [
3      0,
4      1,
5      4,
6      9,
7      16,
8      25,
9      36,
10     49,
11     64,
12     81,
13     100,
14     121,
15     144,
16     169,
17     196,
18     225,
19     256,
20     289,
21     324,
22     361,
23     400,
24     441,
25     484,
26     529,
```

GET



http://127.0.0.1:8000/api15/run15/

Params

Auth

Headers (6)

Body

Pre-req.

Tests

Sett

Body

Pretty

Raw

Preview

Visualize

JSON



26	529,
27	576,
28	625,
29	676,
30	729,
31	784,
32	841,
33	900,
34	961,
35	1024,
36	1089,
37	1156,
38	1225,
39	1296,
40	1369,
41	1444,
42	1521,
43	1600,
44	1681,
45	1764,
46	1849,
47	1936,
48	2025,
49	2116,
50	2209,
51	2304,

GET

▼

http://127.0.0.1:8000/a

Params

Auth

Headers (6)

Body

Pre

body

▼

Pretty

Raw

Preview

Visualiz

51	2304,	76	5329,
52	2401,	77	5476,
53	2500,	78	5625,
54	2601,	79	5776,
55	2704,	80	5929,
56	2809,	81	6084,
57	2916,	82	6241,
58	3025,	83	6400,
59	3136,	84	6561,
60	3249,	85	6724,
61	3364,	86	6889,
62	3481,	87	7056,
63	3600,	88	7225,
64	3721,	89	7396,
65	3844,	90	7569,
66	3969,	91	7744,
67	4096,	92	7921,
68	4225,	93	8100,
69	4356,	94	8281,
70	4489,	95	8464,
71	4624,	96	8649,
72	4761,	97	8836,
73	4900,	98	9025,
74	5041,	99	9216,
75	5184,	100	9409,
76	5329,	101	9604,

در بالا به بررسی کدها و سناریو های تمرین پرداختیم توضیحات تکمیلی :

در هر کد با کمک fastapi نوشته شد با استفاده از ماژول روتر بصورت بسط کردن بکار گرفته میشود که در main مشخص است بصورتی که پس از نوشتن هر کد در main شی از روتر را می سازد و با استفاده از آن به دنبال اجرای آن با APIRouter است

```
from fastapi import FastAPI
from fastapi.routing import APIRouter
from main1_1 import router as main1_1_router
from main1_2 import router as main1_2_router
from main1_3 import router as main1_3_router
from main1_4 import router as main1_4_router
from main1_5 import router as main1_5_router
from main1_6 import router as main1_6_router
from main1_7 import router as main1_7_router

from main2_1 import router as main2_1_router
from main2_2 import router as main2_2_router
from main2_3 import router as main2_3_router
from main2_4 import router as main2_4_router
from main2_5 import router as main2_5_router
from main2_6 import router as main2_6_router
from main2_7 import router as main2_7_router
from main2_8 import router as main2_8_router

app = FastAPI()
app.include_router(main1_1_router, prefix="/api1")
app.include_router(main1_2_router, prefix="/api2")
app.include_router(main1_3_router, prefix="/api3")
app.include_router(main1_4_router, prefix="/api4")
app.include_router(main1_5_router, prefix="/api5")
```

```

23
24 app = FastAPI()
25 app.include_router(main1_1_router, prefix="/api1")
26 app.include_router(main1_2_router, prefix="/api2")
27 app.include_router(main1_3_router, prefix="/api3")
28 app.include_router(main1_4_router, prefix="/api4")
29 app.include_router(main1_5_router, prefix="/api5")
30 app.include_router(main1_6_router, prefix="/api6")
31 app.include_router(main1_7_router, prefix="/api7")
32
33 app.include_router(main2_1_router, prefix="/api8")
34 app.include_router(main2_2_router, prefix="/api9")
35 app.include_router(main2_3_router, prefix="/api10")
36 app.include_router(main2_4_router, prefix="/api11")
37 app.include_router(main2_5_router, prefix="/api12")
38 app.include_router(main2_6_router, prefix="/api13")
39 app.include_router(main2_7_router, prefix="/api14")
40 app.include_router(main2_8_router, prefix="/api15")
41
42
43 if __name__ == "__main__":
44     import uvicorn
45     uvicorn.run(app, host="0.0.0.1", port=8000)
46

```

روند اجرای پروژه :

۱. نوشتن هر سناریو

۲. داکرایز کردن و ساختن image از هر کد با نوشتن Dockerfile, docker-compose.yml که محتوای هر کدام در فایل هست

سپس با کد :


```
nooshin@DESKTOP-A49LPF5: ~/pp.fainal
[+] Building 73.8s (11/11) FINISHED
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 272B 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/library/python:latest 0.0s
=> [1/6] FROM docker.io/library/python:latest 0.0s
=> [internal] load build context 0.9s
=> => transferring context: 47.76MB 0.9s
=> CACHED [2/6] WORKDIR /app 0.0s
=> [3/6] COPY . /app 0.8s
=> [4/6] COPY requirements.txt . 0.1s
=> [5/6] RUN pip install -U pip 10.7s
=> [6/6] RUN pip install -r requirements.txt 60.2s
=> exporting to image 0.9s
=> => exporting layers 0.8s
=> => writing image sha256:b39ff5eda54c3b2d4fd09df538f48ab6b8c00d022099f0 0.0s
=> => naming to docker.io/library/ppfainal-app 0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
[+] Running 1/1
  Network ppfainal/main Created 0.6s
  Container pp-fainal Creating 0.0s
Error response from daemon: Conflict. The container name "/pp-fainal" is already in use by container "fb4cc6aff4a9fc44105b606ad95510b7f91824da6e2a7bd30bd8d7dfd898abf3". You have to remove (or rename) that container to be able to reuse that name.
nooshin@DESKTOP-A49LPF5:~/pp.fainal$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
fb4cc6aff4a9   pp_fainal-app "uvicorn --host 0.0.0..." 2 days ago    Up 9 minutes  0.0.0.0:8000->8000/tcp             pp-fainal
nooshin@DESKTOP-A49LPF5:~/pp.fainal$ docker-compose up -d
[+] Running 1/1
  Container pp.fainal Started 1.1s
nooshin@DESKTOP-A49LPF5:~/pp.fainal$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
cd685dcfd65b   ppfainal-app  "uvicorn --host 0.0.0..." 15 seconds ago Up 14 seconds  0.0.0.0:8000->8000/tcp             pp.fainal
fb4cc6aff4a9   pp_fainal-app "uvicorn --host 0.0.0..." 2 days ago    Exited (0) 59 seconds ago         pp-fainal
nooshin@DESKTOP-A49LPF5:~/pp.fainal$ docker stop pp-fainal
pp-fainal
nooshin@DESKTOP-A49LPF5:~/pp.fainal$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
cd685dcfd65b   ppfainal-app  "uvicorn --host 0.0.0..." About a minute ago Up About a minute  0.0.0.0:8000->8000/tcp             pp.fainal
fb4cc6aff4a9   pp_fainal-app "uvicorn --host 0.0.0..." 2 days ago    Exited (0) 2 minutes ago         pp-fainal
nooshin@DESKTOP-A49LPF5:~/pp.fainal$
```

به ایجاد کانترینرها پرداختیم

و در آخر:

با ابزار Gitbash به push کردن پروژه در گیت هاب پرداختیم:

۱. git init

۲. git add .

۳. git commit -m "hello"

۴. git remote add origin <http://github.com/nooshinnozariii/paralellfainal>

۵. git push -u origin truck

nooshinnozarii/parallelfainal x FastAPI - Swagger UI x Codespaces x Welcome - parallelfainal [Codes x Fas

github.com/nooshinnozarii/parallelfainal

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

parallelfainal Public Pin Unwatch 1

truck 1 Branch 0 Tags Go to file Add file <> Code

nooshinnozarii Delete ppfainal1 directory 7e82eea · 18 hours ago 3 Commits

.idea	hello	19 hours ago
__pycache__	hello	19 hours ago
Dockerfile	hello	19 hours ago
docker-compose.yml	hello	19 hours ago
main.py	hello	19 hours ago
main1_1.py	hello	19 hours ago
main1_2.py	hello	19 hours ago
main1_3.py	hello	19 hours ago
main1_4.py	hello	19 hours ago
main1_5.py	hello	19 hours ago
main1_6.py	hello	19 hours ago
main1_7.py	hello	19 hours ago
main2_1.py	hello	19 hours ago
main2_2.py	hello	19 hours ago

Links Microsoft Edge


در نهایت برای نمایش عمومی بر روی سرور:

وارد codespace در گیت هاب میشویم سپس وارد اکانت گیت هاب خود در codespace میشویم و با مراحل:


Create a new codespace

Repository
To be cloned into your codespace

nooshinnozariii/p... ▾

 Codespace usage for this repository is paid for by nooshinnozariii.

Branch
This branch will be checked out on creation

 truck ▾

Region
Your codespace will run in the selected region

Southeast Asia ▾

Machine type
Resources for your codespace

2-core ▾

Create codespace



Rec 0001.mp4