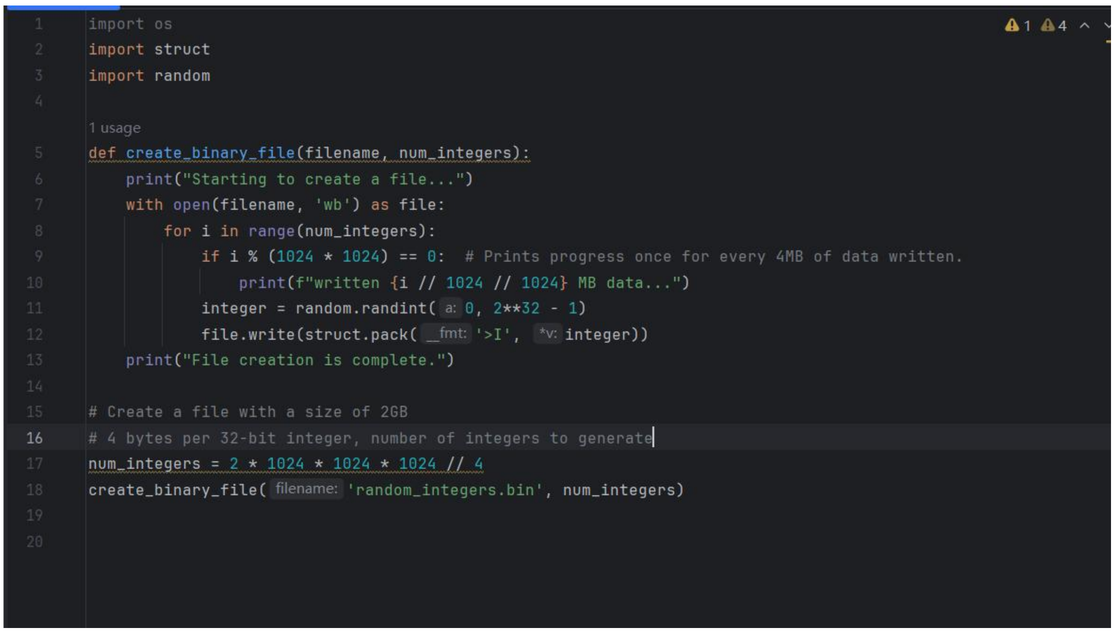
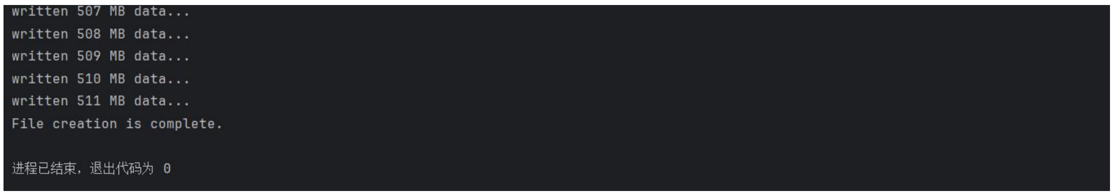
Задание 1

ХУАН ЦЗИНЬЯНЬ

Creates a binary file:

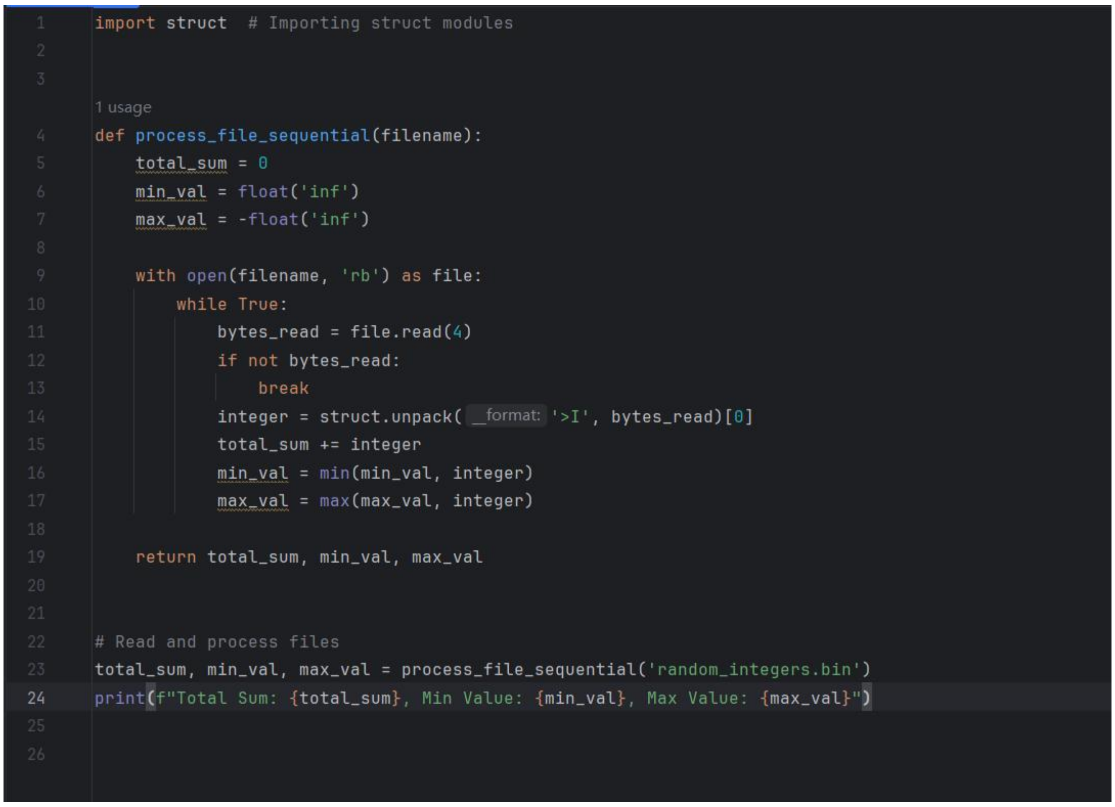


```python
import os
import struct
import random

1 usage
def create_binary_file(filename, num_integers):
    print("Starting to create a file...")
    with open(filename, 'wb') as file:
        for i in range(num_integers):
            if i % (1024 * 1024) == 0:  # Prints progress once for every 4MB of data written.
                print(f"written {i // 1024 // 1024} MB data...")
            integer = random.randint( a: 0, 2**32 - 1)
            file.write(struct.pack( __fmt: '>I',  *v: integer))
    print("File creation is complete.")

# Create a file with a size of 2GB
# 4 bytes per 32-bit integer, number of integers to generate
num_integers = 2 * 1024 * 1024 * 1024 // 4
create_binary_file( filename: 'random_integers.bin', num_integers)
```

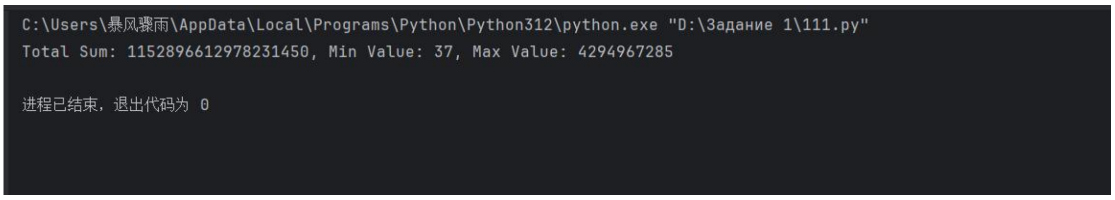


```
written 507 MB data...
written 508 MB data...
written 509 MB data...
written 510 MB data...
written 511 MB data...
File creation is complete.

进程已结束，退出代码为 0
```

Read and process binary files：

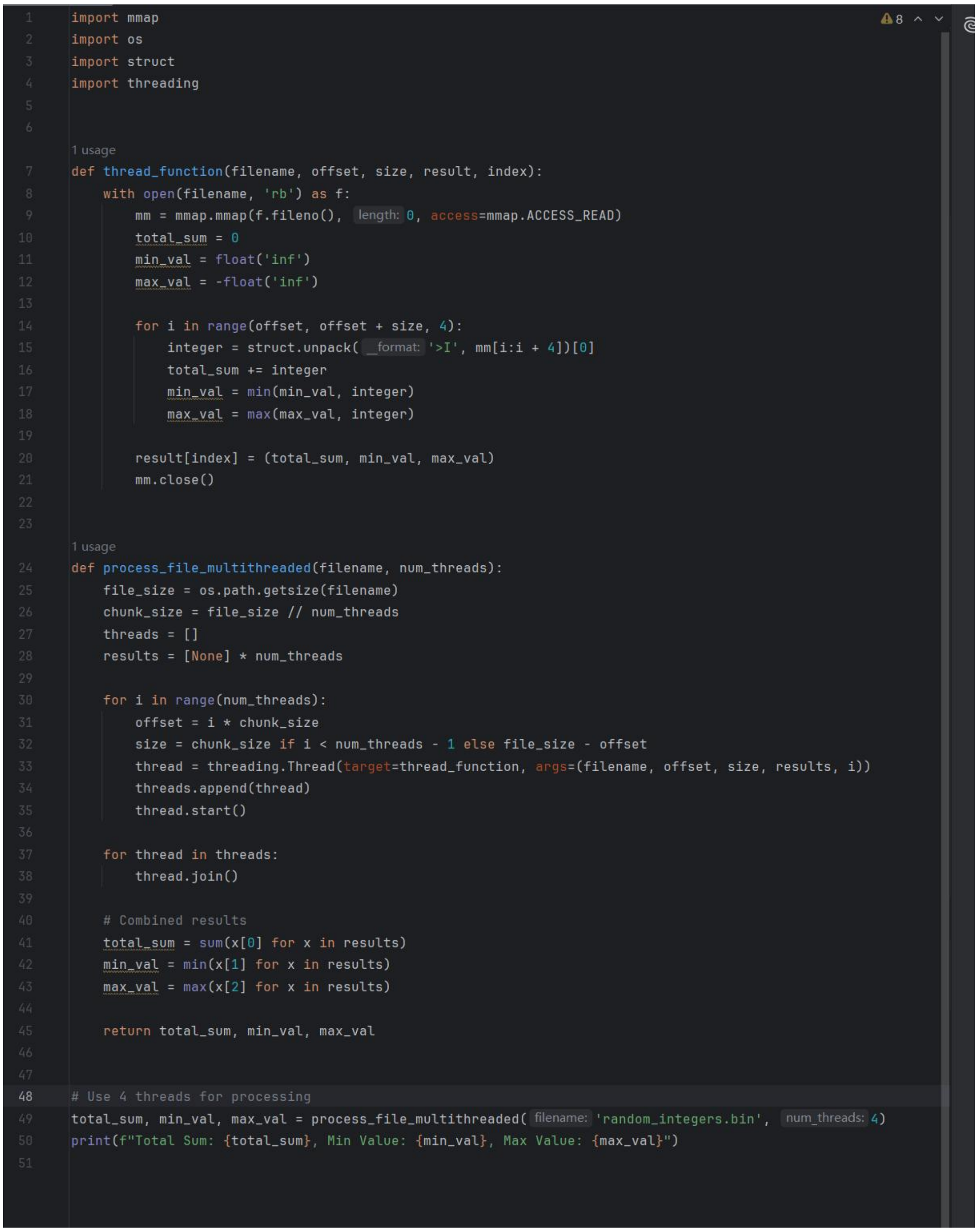


```
import struct  # Importing struct modules


1 usage
def process_file_sequential(filename):
    total_sum = 0
    min_val = float('inf')
    max_val = -float('inf')

    with open(filename, 'rb') as file:
        while True:
            bytes_read = file.read(4)
            if not bytes_read:
                break
            integer = struct.unpack( __format: '>I', bytes_read)[0]
            total_sum += integer
            min_val = min(min_val, integer)
            max_val = max(max_val, integer)

    return total_sum, min_val, max_val


# Read and process files
total_sum, min_val, max_val = process_file_sequential('random_integers.bin')
print(f"Total Sum: {total_sum}, Min Value: {min_val}, Max Value: {max_val}")
```

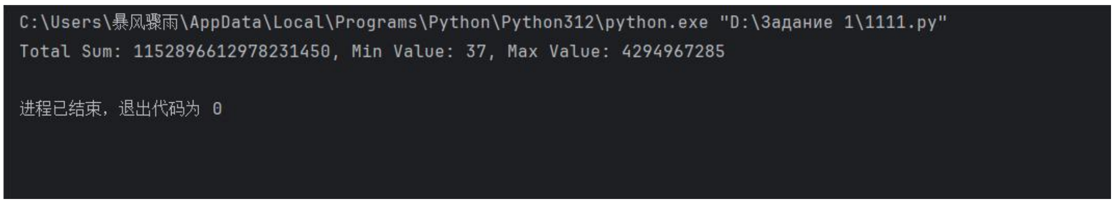


```
C:\Users\暴风骤雨\AppData\Local\Programs\Python\Python312\python.exe "D:\Задание 1\111.py"
Total Sum: 1152896612978231450, Min Value: 37, Max Value: 4294967285

进程已结束，退出代码为 0
```

For the implementation of multithreading and memory mapped files, I use Python's mmap module to map files and the threading module for multithreaded processing.



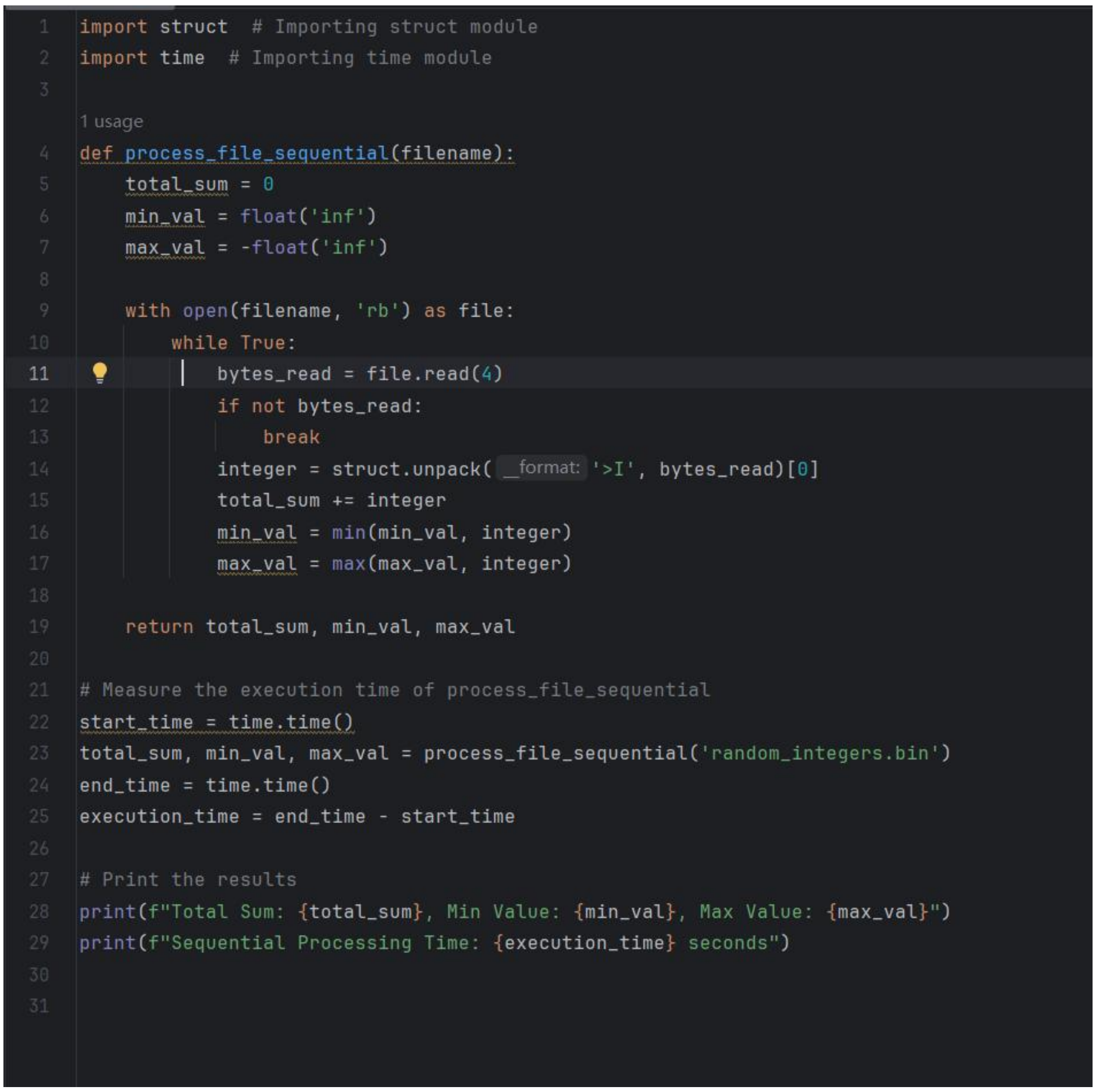
```python
import mmap
import os
import struct
import threading


def thread_function(filename, offset, size, result, index):
    with open(filename, 'rb') as f:
        mm = mmap.mmap(f.fileno(), 0, access=mmap.ACCESS_READ)
        total_sum = 0
        min_val = float('inf')
        max_val = -float('inf')

        for i in range(offset, offset + size, 4):
            integer = struct.unpack('>I', mm[i:i + 4])[0]
            total_sum += integer
            min_val = min(min_val, integer)
            max_val = max(max_val, integer)

        result[index] = (total_sum, min_val, max_val)
        mm.close()


def process_file_multithreaded(filename, num_threads):
    file_size = os.path.getsize(filename)
    chunk_size = file_size // num_threads
    threads = []
    results = [None] * num_threads

    for i in range(num_threads):
        offset = i * chunk_size
        size = chunk_size if i < num_threads - 1 else file_size - offset
        thread = threading.Thread(target=thread_function, args=(filename, offset, size, results, i))
        threads.append(thread)
        thread.start()

    for thread in threads:
        thread.join()

    # Combined results
    total_sum = sum(x[0] for x in results)
    min_val = min(x[1] for x in results)
    max_val = max(x[2] for x in results)

    return total_sum, min_val, max_val


# Use 4 threads for processing
total_sum, min_val, max_val = process_file_multithreaded('random_integers.bin', 4)
print(f"Total Sum: {total_sum}, Min Value: {min_val}, Max Value: {max_val}")
```



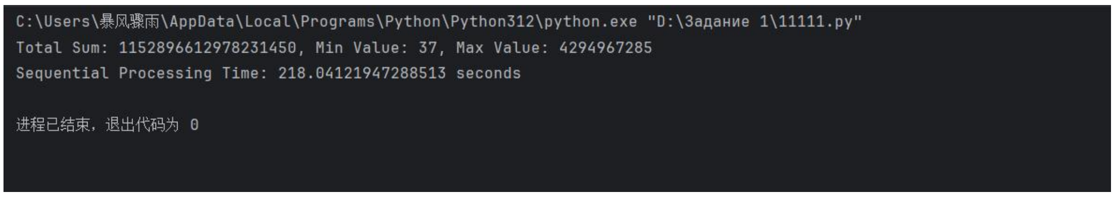
```
C:\Users\暴风骤雨\AppData\Local\Programs\Python\Python312\python.exe "D:\Задание 1\1111.py"
Total Sum: 1152896612978231450, Min Value: 37, Max Value: 4294967285

进程已结束，退出代码为 0
```

Runtime Comparison:

1. Sequential processing time

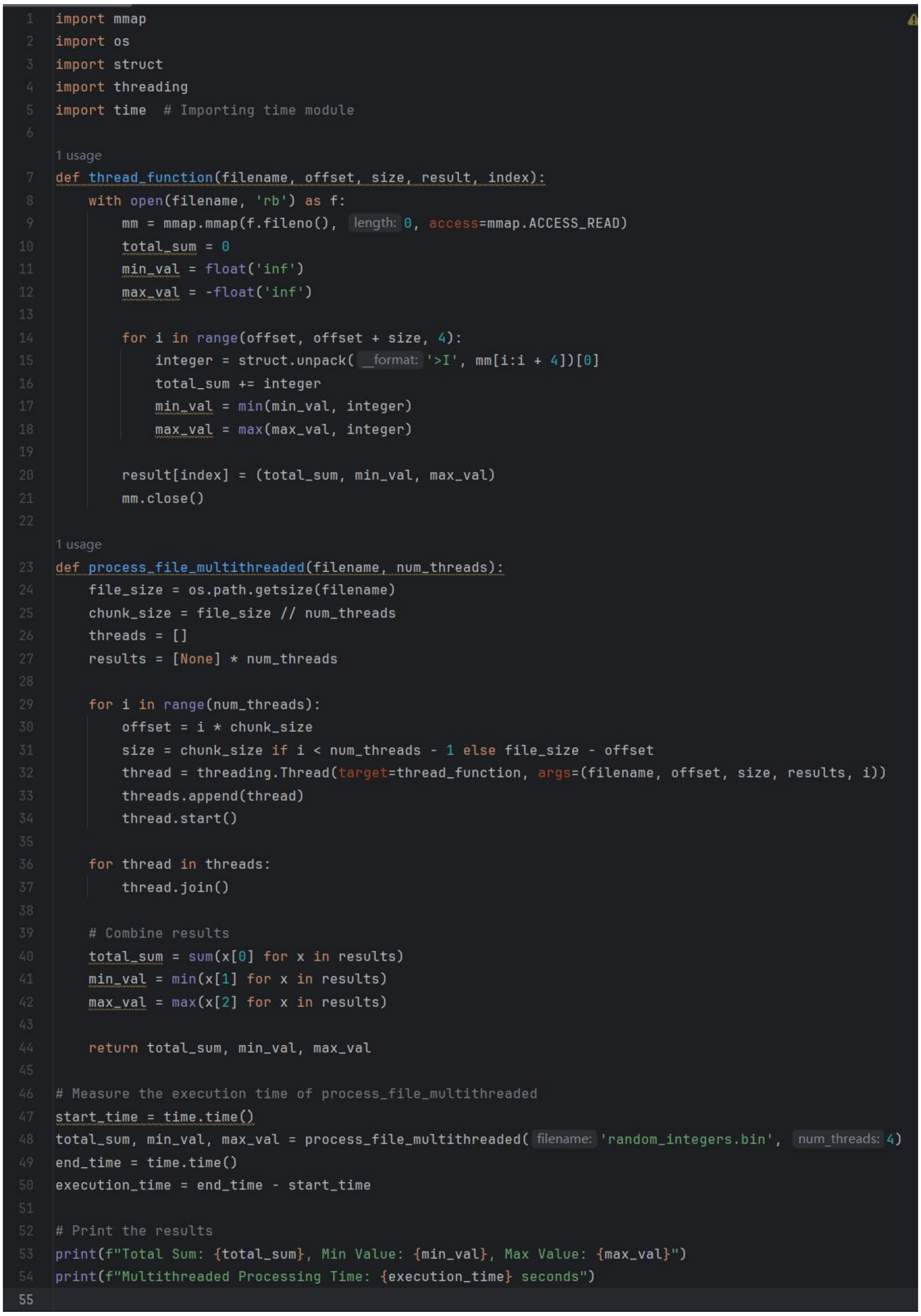


```python
import struct  # Importing struct module
import time  # Importing time module

1 usage
def process_file_sequential(filename):
    total_sum = 0
    min_val = float('inf')
    max_val = -float('inf')

    with open(filename, 'rb') as file:
        while True:
            bytes_read = file.read(4)
            if not bytes_read:
                break
            integer = struct.unpack( __format: '>I', bytes_read)[0]
            total_sum += integer
            min_val = min(min_val, integer)
            max_val = max(max_val, integer)

    return total_sum, min_val, max_val

# Measure the execution time of process_file_sequential
start_time = time.time()
total_sum, min_val, max_val = process_file_sequential('random_integers.bin')
end_time = time.time()
execution_time = end_time - start_time

# Print the results
print(f"Total Sum: {total_sum}, Min Value: {min_val}, Max Value: {max_val}")
print(f"Sequential Processing Time: {execution_time} seconds")
```

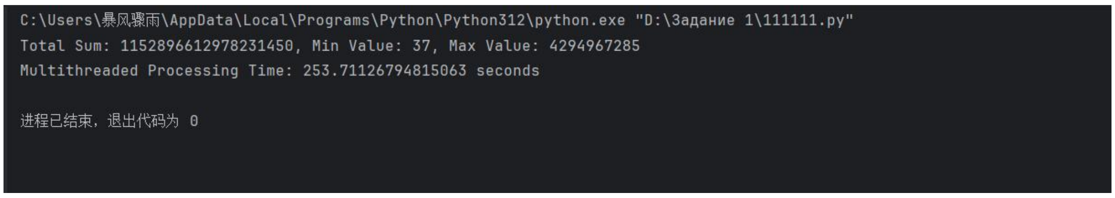


```
C:\Users\暴风骤雨\AppData\Local\Programs\Python\Python312\python.exe "D:\Задание 1\11111.py"
Total Sum: 1152896612978231450, Min Value: 37, Max Value: 4294967285
Sequential Processing Time: 218.04121947288513 seconds

进程已结束，退出代码为 0
```

2.Multi-threaded processing time：



```python
import mmap
import os
import struct
import threading
import time  # Importing time module

1 usage
def thread_function(filename, offset, size, result, index):
    with open(filename, 'rb') as f:
        mm = mmap.mmap(f.fileno(),  length: 0, access=mmap.ACCESS_READ)
        total_sum = 0
        min_val = float('inf')
        max_val = -float('inf')

        for i in range(offset, offset + size, 4):
            integer = struct.unpack( __format: '>I', mm[i:i + 4])[0]
            total_sum += integer
            min_val = min(min_val, integer)
            max_val = max(max_val, integer)

        result[index] = (total_sum, min_val, max_val)
        mm.close()

1 usage
def process_file_multithreaded(filename, num_threads):
    file_size = os.path.getsize(filename)
    chunk_size = file_size // num_threads
    threads = []
    results = [None] * num_threads

    for i in range(num_threads):
        offset = i * chunk_size
        size = chunk_size if i < num_threads - 1 else file_size - offset
        thread = threading.Thread(target=thread_function, args=(filename, offset, size, results, i))
        threads.append(thread)
        thread.start()

    for thread in threads:
        thread.join()

    # Combine results
    total_sum = sum(x[0] for x in results)
    min_val = min(x[1] for x in results)
    max_val = max(x[2] for x in results)

    return total_sum, min_val, max_val

# Measure the execution time of process_file_multithreaded
start_time = time.time()
total_sum, min_val, max_val = process_file_multithreaded( filename: 'random_integers.bin',  num_threads: 4)
end_time = time.time()
execution_time = end_time - start_time

# Print the results
print(f"Total Sum: {total_sum}, Min Value: {min_val}, Max Value: {max_val}")
print(f"Multithreaded Processing Time: {execution_time} seconds")
```



```
C:\Users\暴风骤雨\AppData\Local\Programs\Python\Python312\python.exe "D:\Задание 1\111111.py"
Total Sum: 1152896612978231450, Min Value: 37, Max Value: 4294967285
Multithreaded Processing Time: 253.71126794815063 seconds

进程已结束，退出代码为 0
```