Документация за Клиент-Сървър приложение, което реализира паралелен SelectionSort

Общ преглед

Това Python приложение представлява клиент-сървър система, при която всеки клиент, свързал се към сървъра, въвежда списък от числа, изпраща го на сървъра за сортиране и получава от сървъра сортирания списък. Използва се многонишков модел, за да се позволи паралелно изпълнение на множество клиенти.

Файлове

- **client.py**: Съдържа кода за страната на клиента.
- **server.py**: Съдържа кода за страната на сървъра.

Документация на Клиента

- ### main function call

  Main функцията е начална точка за изпълнението на клиента. Тя създава цикъл, който продължава да се изпълнява докато потребителят въвежда текст, различен от "stop". Всеки път, когато потребителят въведе текст различен от "stop", се изпълнява функцията `handle_user()`. Накрая, кодът отпечатва "Exiting!" след като потребителят въведе "stop" и цикълът приключи.

```
if __name__ == "__main__":
user_input = ""
while user_input.lower() != "stop":
    user_input = input("Enter 'stop' to stop the client execution, 'no' for query: ")
    if user_input.lower() != "stop":
        handle_user()

print("Exiting!")
```

- ### handle_user()

  Функцията `handle_user()` извършва следните действия:

1. Потребителят се подканва да въведе броя на елементите, които желае да сортира.
2. Използвайки цикъл `for`, програмата чака въвеждането на стойности на нов ред за всеки елемент от потребителя и ги добавя в списъка `unsorted_list`.
3. След успешното въвеждане на всички елементи, програмата отпечатва несортирания списък на екрана.
4. След това функцията извиква друга функция, наречена `send_data(unsorted_list)`, която ще изпрати несортирания списък към сървъра.

```
def handle_user():
unsorted_list = []
n = int(input("Enter number of elements you want to sort: "))
for i in range (0, n):
    elem = int(input())
    unsorted_list.append(elem)

print(f"Sending unsorted list to server: {unsorted_list}")
send_data(unsorted_list)
```

- ### send_data(data)

  Функцията `send_data(data)` извършва следните действия:

1. Създава сокет за клиента с IPv4 адресиране и използване на TCP протокол.
2. Установява връзка със сървъра, който се намира на локалния хост 127.0.0.1 и слуша на порт 8888.
3. Преобразува списъка от числа `data` в низ (за да може да се кодира), като числата се разделят със запетаи. Този низ се кодира в байтов формат и се изпраща на сървъра чрез метода `send` на клиентския сокет.
4. Чака отговор от сървъра чрез приемане на данни с размер до 1024 байта. Декодира получените данни от сървъра, които представляват сортирания списък в низ.
5. Разделя низа в списък от числа, използвайки запетаите като разделител.
6. Отпечатва на екрана сортирания списък, който е получен от сървъра.
7. Затваря връзката със сървъра чрез затварянето на клиентския сокет. **Ако възнине някаква грешка (напр. клиентът не може да се свърже към сървъра по някаква причина, то тя ще бъде уловена от Exception и изкарана на екрана на клиента!**

```
def send_data(data):
  try:
    #open a socket for the client choosing IPv4 with TCP
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect(('127.0.0.1', 8888))

    data_str = ','.join(map(str, data))
    client.send(data_str.encode())

    sorted_data = client.recv(1024).decode()
    sorted_numbers = [int(num) for num in sorted_data.split(',')]

    print(f"Received sorted list from server: {sorted_numbers}")

    client.close()

  except Exception as e:
   print(f"Can't connect to the server. Error: {e}")
   sys.exit()
```

Demo на клиента:

```
marto:~/Desktop/ClientServerApp/working_v2$ python3 client.py
Enter 'stop' to stop the client execution, 'no' to continue: no
Enter number of elements you want to sort: 3
8
12
1
Sending unsorted list to server: [8, 12, 1]
Can't connect to the server. Error: [Errno 111] Connection refused
marto:~/Desktop/ClientServerApp/working_v2$ python3 client.py
Enter 'stop' to stop the client execution, 'no' to continue: no
Enter number of elements you want to sort: 5
16
28
1
13
25
Sending unsorted list to server: [16, 28, 1, 13, 25]
Received sorted list from server: [1, 13, 16, 25, 28]
Enter 'stop' to stop the client execution, 'no' to continue: no
Enter number of elements you want to sort: 4
2
18
6
1
Sending unsorted list to server: [2, 18, 6, 1]
Received sorted list from server: [1, 2, 6, 18]
Enter 'stop' to stop the client execution, 'no' to continue: stop
Exiting!
```

Документация на Сървъра

- ### main function call

  Main функцията е начална точка за изпълнението на сървъра. В случая на този код блокът, който се изпълнява, е `start_server()`, която стартира сървъра.

  ```
  if __name__ == "__main__":
      start_server()
  ```

- ### start_server()

  `start_server()` е функцията, която съдържа основната логика за стартиране на сървъра. Ето обяснение на нейните основни етапи:

1. **Създаване на сървърен сокет**:

   ```
   server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  server.bind(('127.0.0.1', 8888))
  server.listen(5)
   ```

   Тук се създава сървърен сокет (`server`), който използва IPv4 адресация (`socket.AF_INET`) и TCP протокол (`socket.SOCK_STREAM`). Сървърът се

свързва към адрес '127.0.0.1' и порт 8888 и започва да слуша за входящи връзки.

2. **Безкрайен цикъл за приемане на връзки:**

```
while True:
    client, addr = server.accept()
    print(f"Accepted connection from {addr[0]}:{addr[1]}")
    client_handler = threading.Thread(target=handle_client, args=(client,))
    client_handler.start()
```

Сървърът влиза в безкраен цикъл, в който изчаква връзки от клиенти чрез `server.accept()`. Когато клиент се свърже, сървърът създава нова нишка (`client_handler`), която изпълнява функцията `handle_client` и подава клиентския сокет като аргумент.

3. **Обработка на клиентската връзка:**

```
def handle_client(client_socket):
    # ... (вижте по-долу)
```

Функцията `handle_client` се изпълнява в нова нишка за всеки свързан клиент. Тя получава несортиран списък от клиента, извършва паралелен selection sort върху него и изпраща сортирания списък обратно на клиента.

4. **Затваряне на сървърния сокет:**

```
server.close()
```

След като сървърът бъде затворен (например, чрез прекъсване на изпълнението на скрипта чрез Ctrl+C), този ред затваря сървърния сокет.

Важно е да се отбележи, че кодът съдържа закоментирани части, които са свързани с проверка за активност и затваряне на сървъра след определен период от бездейност (т.е не получава заявки да сортира списък за определен период от време). Тези части обаче са закоментирани, защото един сървър е хубаво да работи постоянно, така че не се изпълняват в момента (експериментирах просто с разни работи :)).

- ## handle_client(client_socket)

`handle_client(client_socket)` е функция, която се изпълнява в отделна нишка за всеки клиент, който се свърже към сървъра.

1. **Получаване на данни от клиента:**

```
data = client_socket.recv(1024)
numbers = [int(num) for num in data.decode().split(',')]
```

Функцията използва `recv(1024)`, за да приеме данни от клиента. Предполага се, че данните се предават чрез мрежовата връзка в части от по 1024 байта. Получените байтове се декодират от байтове в символи, след което се разделят по символа ',' и се преобразуват в списък от цели числа.

2. **Печат на несортиран списък:**

```
print(f"Received unsorted list: {numbers}")
```

Функцията извежда несортирания списък, който е получен от клиента, на конзолата на сървъра.

3. **Единична сортировка с една нишка:**

```
start_time_single = time.time()
selection_sort(numbers, 0, len(numbers))
end_time_single = time.time()
elapsed_time_single = end_time_single - start_time_single
print(f"Single-threaded selection sort took {elapsed_time_single:.6f} seconds.")
```

Списъкът се сортира със selection sort алгоритъм, използвайки само една нишка. Започва се засичане на времето преди и след сортирането, за да се определи колко време отнема този процес.

4. **Възстановяване на оригиналния списък:**

```
numbers = numbersCpy.copy()
print(f"Reset the list to original: {numbers}")
```

След еднонишковото сортиране списъкът се възстановява до оригиналната си версия.

5. **Паралелен selection sort с 2 нишки:**

```
multi_threaded_selection_sort(numbers)
```

Извиква функцията `multi_threaded_selection_sort`, която изпълнява паралелен selection sort върху подадения списък.

6. **Преобразуване на сортирания списък във формат, подходящ за изпращане:**

```
sorted_data = ','.join(map(str, numbers))
```

Сортираният списък се обработва така, че да бъде представен като един символен низ, в който всеки елемент е разделен от следващия със запетая.

7. **Изпращане на сортирания списък на клиента:**

```
client_socket.send(sorted_data.encode())
```

Сортираният списък се изпраща към клиента след като се кодира в байтов формат (`.encode()`).

8. **Затваряне на клиентския сокет:**

```
client_socket.close()
```

Клиентският сокет се затваря, тъй като вече са обработени и изпратени данните на клиента.

9. **Обработка на грешки:**

```
except Exception as e:
    print(f"Error handling client: {e}")
```

Ако възникне проблем при обработката на данните на клиента, съобщението за грешка се извежда на конзолата. Това предпазва от прекъсване на изпълнението на целия сървър поради проблем с един клиент.

- ## multi_threaded_selection_sort(arr, num_threads=2)

```
 def multi_threaded_selection_sort(arr, num_threads=2):
segment_size = len(arr) // num_threads
threads = []

start_time = time.time()

for i in range(num_threads):
    start = i * segment_size
    end = (i + 1) * segment_size if i != num_threads - 1 else len(arr)
    thread = threading.Thread(target=selection_sort, args=(arr, start, end))
    threads.append(thread)
    thread.start()

for thread in threads:
    thread.join()

mid  = (num_threads  -  1) *  segment_size
merge(arr, 0, mid, len(arr))

end_time = time.time()
elapsed_time = end_time - start_time
print(f"Multi-threaded selection sort took {elapsed_time:.6f} seconds.")
```

1. `segment_size = len(arr) // num_threads` : Разделяме дължината на масива (`arr`) на броя на нишките (`num_threads=2`), за да определим размера на всеки сегмент от масива, който всяка нишка ще сортира.
2. `threads = []` : Създаваме празен списък, в който ще съхраняваме обекти от тип нишки.
3. `start_time = time.time()` : Започваме да измерваме времето преди стартирането на сортирането (трябва ни, за да сравним за колко време ще се изпълни паралелен selection sort).
4. `for i in range(num_threads):` : Започваме цикъл, който създава и стартира нишки за всяка част от масива.
    - `start = i * segment_size` : Определя началния индекс на текущия сегмент.
    - `end = (i + 1) * segment_size if i != num_threads - 1 else len(arr)` : Определя краен индекс на текущия сегмент. Ако сме на последната нишка, краен индекс е дължината на масива, в противен случай е крайния индекс на сегмента.
    - `thread = threading.Thread(target=selection_sort, args=(arr, start, end))` : Създаваме обект от тип нишка, като указваме `target` да бъде функцията `selection_sort` , която ще сортира текущия сегмент, и подаваме аргументите ѝ чрез `args` .
    - `threads.append(thread)` : Добавяме обекта от тип нишката към списъка `threads` .
    - `thread.start()` : Стартираме нишката.
5. `for thread in threads:` : Проверяваме дали всяка нишка приключва своята работа.
    - `thread.join()` : Изчакваме всяка нишка да завърши своето изпълнение. Когато използваме `thread.join()` в цикъл, както е представено в кода, програмата ще изчака завършването на всички нишки, преди да продължи към следващите операции. Това е необходимо, защото искаме да сме сигурни, че всички сортировки по сегменти са приключили, преди да продължим с измерването на времето и извеждането на резултата.

6. `merge(arr, 0, mid, len(arr))` : Сливаме двата сортирани сегмента във финалния списък.
7. `end_time = time.time()` : Завършваме измерването на времето след като всички нишки са приключили.
8. `elapsed_time = end_time - start_time` : Изчисляваме общото време, което е изминало от стартирането на сортирането до неговото приключване.
9. `print(f"Multi-threaded selection sort took {elapsed_time:.6f} seconds.")` : Извеждаме времето, което е отнела многонишковата сортировка, с точност до 6 знака след десетичната запетая.

- ## merge(arr, start, mid, end)

  Функцията `merge` има за цел да слива два сортирани сегмента на масива в рамките на същия масив. Това се извършва чрез обединяване на двата сегмента в един сортиран сегмент.

1. `arr` : Самият масив, върху който се извършва сливането.
2. `start` : Индексът на началото на първия сегмент, който трябва да се слее.
3. `mid` : Индексът на средата на масива и края на първия сегмент.
4. `end` : Индексът на края на втория сегмент.

Функцията работи по следния начин:

1. Създава два подмасива - `left` и `right`, като `left` е частта от масива от началото до средата, а `right` е частта от средата до края.
2. Извършва сливане на `left` и `right` в рамките на оригиналния масив `arr`. Това става чрез сравняване на елементите от `left` и `right` и добавяне на по-малкия елемент към `arr`.
3. Когато един от подмасивите се изчерпи, останалите елементи от другия подмасив се добавят към края на `arr`.
4. Крайният резултат е сортиран сегмент на масива, който съдържа всички елементи от обединението на двата сегмента.

- ## selection_sort(arr, start, end)

1. **Избор на минимален елемент:**

   ```
   min_index = i
   for j in range(i + 1, len(arr)):
       if arr[j] < arr[min_index]:
           min_index = j
   ```

   Функцията стартира от индекс `i` и търси минималния елемент в частта на списъка от индекс `i` нататък. Ако намери елемент, по-малък от текущия минимум, обновява `min_index` с новия индекс на минимума.

2. **Размяна на елементите:**

   ```
   arr[i], arr[min_index] = arr[min_index], arr[i]
   ```

   След като бъде намерен минималният елемент, той се разменя с елемента на позиция `i`. Така минималният елемент се поставя на правилната позиция в сортирания подсписък.

Demo на сървъра:

```
marto:~/Desktop/ClientServerApp/working_v2$ python3 server.py
Server listening on port 8888...
Accepted connection from 127.0.0.1:47208
Received unsorted list: [23, 2, 33]
Single-threaded selection sort took 0.000032 seconds.
Reset the list to original: [23, 2, 33]
Multi-threaded selection sort took 0.001506 seconds.
Accepted connection from 127.0.0.1:46676
Received unsorted list: [4, 2, 6, 9, 1]
Single-threaded selection sort took 0.000029 seconds.
Reset the list to original: [4, 2, 6, 9, 1]
Multi-threaded selection sort took 0.001062 seconds.
```

Защо паралелният selection sort почти винаги е по-бавен от този, който се изпълнява на 1 нишка

Многонишковото програмиране в този случай може не винаги да доведе до по-бързи резултати, особено при използването на езици като Python, където има Global Interpreter Lock (GIL). GIL предпазва общите данни от конкурентни модификации и прави трудно паралелното изпълнение на някои операции. Това може да доведе до по-малка ефективност и дори до по-голямо време за изпълнение, отколкото ако използвате една нишка.

- Полезни ресурси, които съм използвал:
  - Какво е GIL?
  - Python Chat Room (за да видя как точно се прави клиент-сървър архитектура на Python)
  - Python Threading Explained