پروژه درس شبکه های کامپیوتری

اعضای گروه:

سارینا نعمتی

لیلا عرفانی راد

بهار۱۴۰۲

بخش اول: ماشین حساب توزیع شده

- سمت server:

- تابع ***perform_calculation*** عملگر، op1 و op2 را به عنوان پارامتر می گیرد و محاسبه درخواستی را بر اساس عملگر انجام می دهد. برای عملیات های حسابی (جمع، تفریق، ضرب، تقسیم)، به سادگی این عملیات را با استفاده از عملوندهای ارائه شده انجام می دهد. برای عملیات مثلثاتی (Sin، Cos، Tan، Cot)، تابع مثلثاتی مربوطه را اعمال می کند. نتیجه محاسبه return می شود.
- تابع ***handle_client_connection*** مسئول پردازش درخواست مشتری و ارسال پاسخ است. سوکت مشتری را به عنوان پارامتر می گیرد. هنگامی که درخواست مشتری دریافت می شود، رمزگشایی شده و با استفاده از جداکننده ***$*** به عملگر، op1 و op2 تقسیم می شود. عملگر مستقیماً در محاسبات استفاده می شود، در حالی که op1 و op2 با استفاده از ***float*** به اعداد ممیز شناور تبدیل می شوند. اگر مقدار ***'op2 'None*** باشد، به این معنی است که اپراتور به عملوند دوم نیاز ندارد، بنابراین ما op2 را روی ***None*** برای رسیدگی به چنین مواردی تنظیم می کنیم.
تابع ***perform_calculation*** با عملگر و عملوندها برای به دست آوردن نتیجه فراخوانی میشود.
response با استفاده از جداکننده ***$***، شامل رشته calculation و result ساخته می‌شود. سپس پاسخ کدگذاری می شود و با استفاده از ***client_socket.send*** به client ارسال می شود. در نهایت سوکت client بسته میشود.

- سمت client:

تابع ***send_calculation_request*** وظیفه ارسال request محاسبه به سرور و دریافت پاسخ را بر عهده دارد. عملگر، op1 و op2 را به عنوان پارامتر می گیرد.
ابتدا، رشته request با استفاده از جداکننده ***$*** برای جداسازی عملگر، op1 و op2 (در صورت وجود) ساخته می‌شود. سپس درخواست با استفاده از ***server_socket.send*** کد گذاری شده و به سرور ارسال می شود.
در مرحله بعد، مشتری منتظر دریافت پاسخ از سرور با استفاده از ***server_socket.recv*** پاسخ رمزگشایی شده و با استفاده از جداکننده ***$*** به بخش ها تقسیم می شود. توجه داشته باشید که اولین و آخرین عناصر پاسخ تقسیم نادیده گرفته می‌شوند، زیرا انتظار می‌رود به دلیل جداکننده‌های ***$*** پیشرو و انتهایی، رشته‌های خالی باشند.
اگر پاسخ شامل حداقل دو بخش باشد، به معنای پاسخ معتبر است. قسمت محاسبه به متغیر ***calculation*** و نتیجه به متغیر ***result*** اختصاص داده می شود. سپس این مقادیر به عنوان خروجی چاپ می شوند.
اگر پاسخ شامل حداقل دو قسمت نباشد، به معنای پاسخ نامعتبر از طرف سرور است و یک پیام خطا چاپ می شود.
در نهایت، سوکت مشتری با استفاده از ***server_socket.close*** بسته می شود.

برای ران کردن این بخش، باید دوتا terminal باز کنیم، در ترمینال اول، server.py را ران کرده و در ترمینال دوم، client.py را ران میکنیم. سپس خروجی ما بصورت زیر میشود:

```
PS E:\SBU\Term8\Shabakeh\HWs\pythonProject\Calculator> python server.py
Server is listening on port 3000...
```

```
PS E:\SBU\Term8\Shabakeh\HWs\pythonProject\Calculator> python client.py

Calculation: 10 Add 2
time:  0:00:00.000001
Result: 12.0

Calculation: 10 Subtract 2
time:  0:00:00.000001
Result: 8.0

Calculation: 10 Multiply 2
time:  0:00:00.000002
Result: 20.0

Calculation: 10 Divide 2
time:  0:00:00.000003
Result: 5.0

Calculation: Sin 30
time:  0:00:00.000004
Result: -0.9880316240928618

Calculation: Cos 30
time:  0:00:00.000004
Result: 0.15425144988758405

Calculation: Tan 30
time:  0:00:00.000004
Result: -6.405331196646276

Calculation: Cot 30
time:  0:00:00.000015
Result: -0.15611995216165922
```

بخش دوم: سیستم توزیع فایل P2P

این کد یک سیستم انتقال فایل peer-to-peer ساده را با استفاده از سوکت های UDP نشان می دهد. می‌تواند در مد سرور برای ارائه فایل‌ها از یک دایرکتوری مشخص یا در مد کلاینت برای دریافت فایل‌ها از سرور اجرا شود. کد اتصالات سوکت لازم را تنظیم می کند، داده های فایل را به صورت chunk ارسال و دریافت می کند و عملیات ورودی/خروجی فایل را مدیریت می کند.

- وارد کردن ماژول های مورد نیاز:
ماژول ***socket*** عملکردهای لازم برای ایجاد و استفاده از سوکت ها برای ارتباطات شبکه را فراهم می کند.
ماژول ***sys*** دسترسی به پارامترها و عملکردهای خاص سیستم را فراهم می کند.
ماژول ***os*** عملکردهایی را برای تعامل با سیستم عامل ارائه می دهد.

- Constant ها:
HOST روی "localhost" تنظیم شده است، که نشان می دهد سرور به رابط localhost متصل می شود.
PORT روی ۳۰۰۰ تنظیم شده است و شماره پورت سوکت سرور را مشخص می کند.
BUFFER_SIZE روی ۱۰۲۴ تنظیم شده است که نشان دهنده حداکثر اندازه بافر داده برای ارسال و دریافت است.

- تابع ***send_file_data***:

این تابع وظیفه ارسال اطلاعات فایل به client را بر عهده دارد. ***File_path*** (مسیر فایل برای ارسال) ، ***client_socket*** (object سوکت سرور) و ***addr*** (آدرس مشتری) را به عنوان پارامتر می گیرد. فایل را در حالت باینری باز می کند و کل اطلاعات فایل را می خواند. از یک لوپ برای iterate روی داده های فایل در chunk های اندازه ***BUFFER_SIZE - 4*** استفاده می کند. برای هر chunk، part no. را به یک offset ۴ بایتی تبدیل می کند و با استفاده از روش ***sendto*** سوکت، قطعه را با offset برای client ارسال می کند. در نهایت، پیامی را چاپ می کند که نشان می دهد داده های فایل ارسال شده است.

- تابع ***server_mode***:

این تابع حالت عملکرد سرور را نشان می دهد. پارامتر ***directory*** را می گیرد که فهرستی که فایل ها در آن قرار دارند را مشخص می کند. با استفاده از ***socket.socket*** با آدرس ***AF_INET FAMILY*** و نوع سوکت ***SOCK_DGRAM*** یک سوکت UDP ایجاد می کند. سوکت سرور را به ***HOST*** و ***PORT*** مشخص شده در constant ها متصل می کند. وارد حلقه ای می شود که در آنجا منتظر دریافت نام فایل از کلاینت به همراه آدرس کلاینت می شود. path فایل کامل را با پیوستن به دایرکتوری مشخص شده و ***file_name*** رمزگشایی شده دریافت شده از کلاینت میسازد. اگر فایل در مسیر مشخص شده وجود نداشته باشد، یک پیغام خطا چاپ می کند و return میکند. در غیر این صورت، تابع ***send_file_data*** را فراخوانی می کند تا داده های فایل را برای کلاینت ارسال کند. در نهایت سوکت سرور را می بندد.

- تابع ***receive_file_data***:

این تابع وظیفه دریافت اطلاعات فایل از سرور را بر عهده دارد. ***Client_socket*** (object سوکت مشتری) و ***offset*** (آخرین .part no دریافت شده) را به عنوان پارامتر می گیرد. یک دیکشنری خالی ***receive_packets*** را برای ذخیره قطعات دریافتی مقداردهی اولیه می کند. از یک for برای iteration در محدوده .part no از 0 تا offset فعلی استفاده می کند. در داخل حلقه، با استفاده از روش ***recvfrom*** سوکت، chunk ای از داده ها را از سرور دریافت میکند. اگر هیچ chunk ای دریافت نشود (که نشان دهنده پایان انتقال داده است)، از for خارج می شود. chunk دریافت شده را decode می کند، .part no و data ها را از chunk استخراج می کند و آنها را در ***receive_packets*** ذخیره میکند. offset را به آخرین .part no دریافت شده به روز می کند. اگر داده های قطعه دریافتی کوچکتر از اندازه بافر باشد، از حلقه خارج می شود. در نهایت، receive_packets و offset به روز شده را return میکند.

- تابع ***client_mode***:

این تابع حالت عملکرد کلاینت را نشان می دهد. ***file_name*** (نام فایلی که باید از سرور دریافت شود) و ***saving_path*** (مسیر ذخیره فایل دریافتی) را به عنوان پارامتر می گیرد. متغیر offset را به 0 مقداردهی میکند. با استفاده از ***socket.socket*** با آدرس ***AF_INET FAMILY*** و نوع سوکت ***SOCK_DGRAM*** یک سوکت UDP ایجاد می کند. نام ***file_name*** را که به صورت byte کدگذاری شده است به سرور در ***HOST*** و ***PORT*** مشخص شده با استفاده از روش ***sendto*** سوکت ارسال میکند. تابع ***receive_file_data*** را call می‌کند تا file data را از سرور دریافت کند، سوکت client و offset فعلی را ارسال کند. با باز کردن فایل ***saving_path*** در mode نوشتن باینری و نوشتن قسمت های دریافتی به order صحیح، داده های فایل دریافتی را ذخیره می کند. اگر هر بخشی miss شده باشد (در بسته های دریافتی یافت نشد)، یک پیام خطا ایجاد می کند. در نهایت، پیامی را چاپ می کند که نشان می دهد داده های فایل دریافت و ذخیره شده است و سوکت مشتری را می بندد.

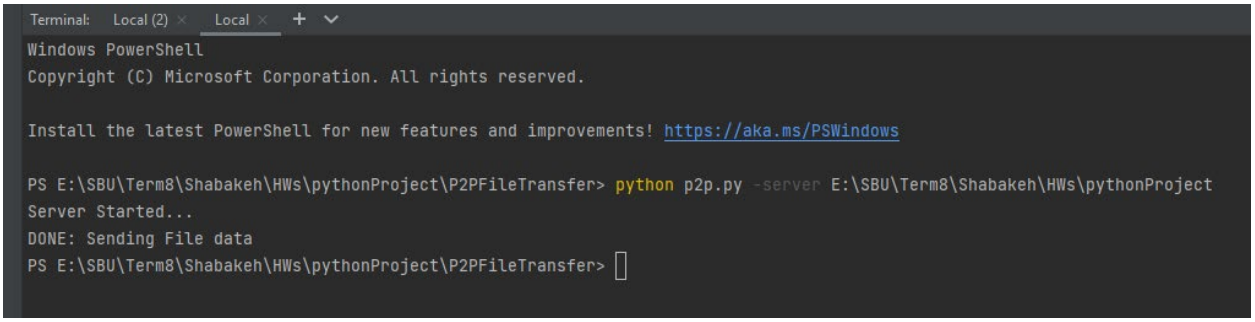
- تابع ***create_dir***:

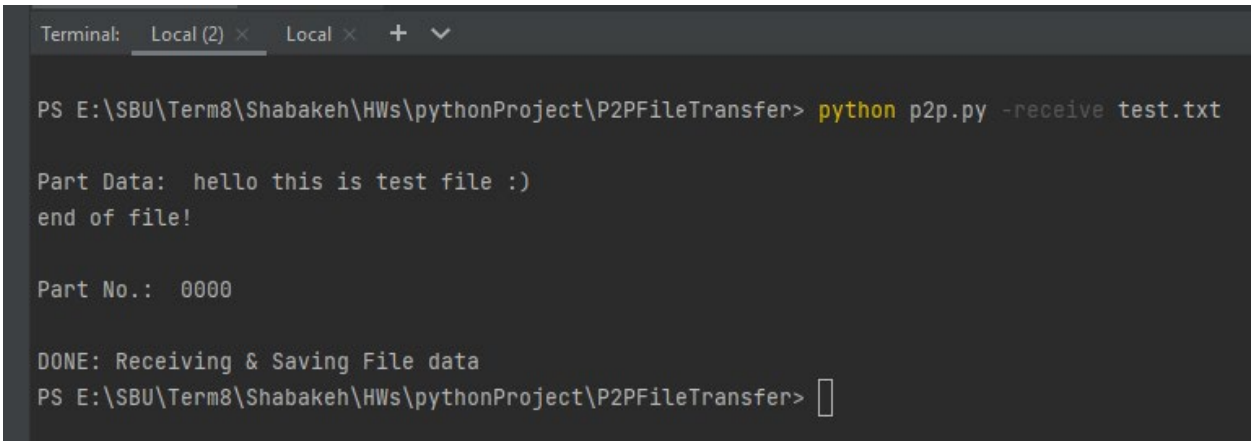
این تابع وظیفه ایجاد directory را بر عهده دارد. پارامتر ***directory*** را می گیرد که مسیر directory را برای ایجاد مشخص می کند. تلاش می کند تا دایرکتوری را با استفاده از ***os.makedirs*** ایجاد کند. اگر در حین ایجاد دایرکتوری خطای ***OSE*** رخ دهد، پیام خطا را چاپ می کند. در غیر این صورت، پیامی را چاپ می کند که نشان می دهد دایرکتوری ایجاد شده است.

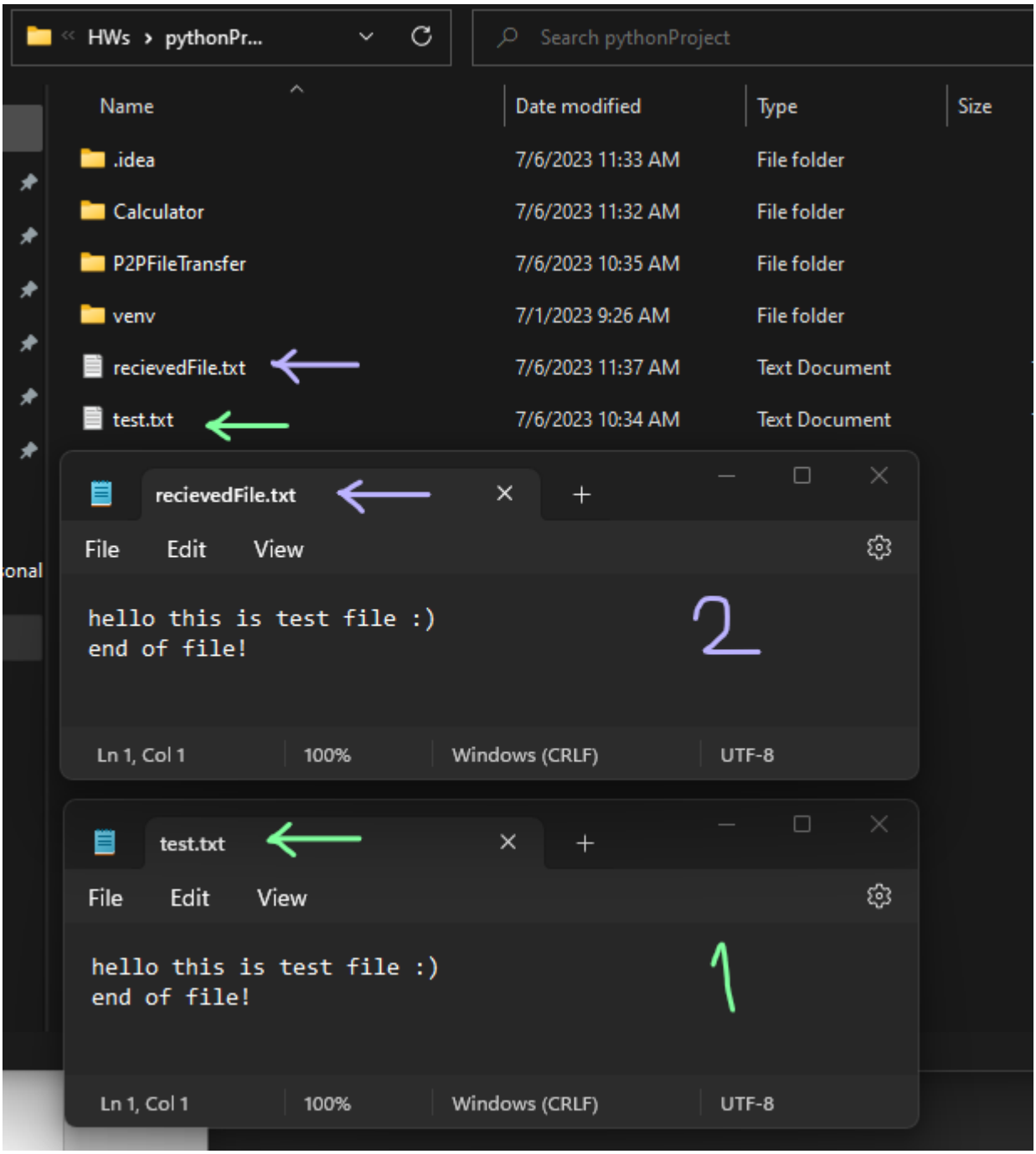
- تابع main:

این تابع نقطه ورود برنامه است.command-line arguments را برای تعیین mode برنامه بررسی می کند. اگر حالت ***server*** باشد، انتظار دارد که یک آرگومان اضافی که فهرستی که فایل ها در آن قرار دارند را مشخص کند. اگر حالت ***receive*** باشد، انتظار دارد که یک آرگومان اضافی که نام فایل را از سرور دریافت کند، مشخص کند. تابع mode مناسب را بر اساس آرگومان های خط فرمان فراخوانی می کند. اگر یک حالت invalid مشخص شده باشد، یک پیام خطا چاپ می کند.

حال برای تست کردن این کد، ۲تا ترمینال باز کرده، یکی برای سرور و یکی برای کلاینت. در نهایت در دایرکتوری داده شده، recievedFile.txt را میتوانیم مشاهده کنیم.






HWs › pythonPr...
Search pythonProject
Name
Date modified
Type
Size
.idea
7/6/2023 11:33 AM
File folder
Calculator
7/6/2023 11:32 AM
File folder
P2PFileTransfer
7/6/2023 10:35 AM
File folder
venv
7/1/2023 9:26 AM
File folder
recievedFile.txt
7/6/2023 11:37 AM
Text Document
test.txt
7/6/2023 10:34 AM
Text Document
recievedFile.txt
File
Edit
View
hello this is test file :)
end of file!
2
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8
test.txt
File
Edit
View
hello this is test file :)
end of file!
1
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8