Assignment 3
The Nimbus 10000[*]

CS 4410, Spring 2018, Cornell University

April 9, 2018

1 Abstract

The *Client/Server Paradigm* is a common model for structuring distributed computing. In this assignment, you will be working on developing a *multi*-client, single-server system where the server accepts connections from multiple clients simultaneously. In our model, we will be simulating a cloud backup system; clients can open a connection with a server and send a file for the server to back up. The caveat: the server has a limited buffer size and might not always be able to receive everything from the clients all at once. Even better, the buffer size can change at any time without warning!

Assignment Project Exam Help

2 Motivation

https://powcoder.com

Whenever you use an Internet-connected device, you are making anywhere from dozens to thousands of network requests *every second* to a remote server that tirelessly handles all the requests, ideally without any degradation in performance. As you have learned, you can use the Transport Layer protocol TCP to ensure that your request will reach your destination. TCP will ensure that the request, however large it is, will be reliably delivered and packets are pieced back together in the correct order. However, what TCP cannot guarantee is what happens on either end, "above" the Transport Layer. In this assignment, we will be simulating a server that can accept requests from multiple clients, but only accepts packets containing data no larger than a set amount. The client knows that there is a limit, but it doesn't know the size of the buffer. The server helps the client out by returning a packet indicating whether or not the last packet was of an acceptable size. Using this information, the client's job is to send requests to the server in an efficient manner.

Add WeChat powcoder

3 Introduction

The assignment has three phases. In **Phase One**, you will become familiar with the basics of socket programming in Python by building a server that handles *a single client* and has *an unbounded buffer size* as well as building the client. In **Phase Two**, you will be adding a bounded buffer size to your server, meaning that you will have to *update your client* to probe for this buffer size. Lastly, in **Phase Three**, you will *upgrade your server* to handle multiple clients.

3.1 Assumption

The project will be restricted to the **CSUG servers and Course VM**, so you should use either of these for your development and testing. **We cannot dedicate staff resources to troubleshooting problems associated with not using the CSUG servers or the Course VM.** Please note that A2 was in Python 2 and A3 is in **Python 3.*.***. Make sure you're calling the right version of python (both are installed on

[*]a slightly spotty cloud backup server operating on port 10000 or higher

both the CSUG servers and the VM). One of the main differences you'll notice between the two is that `print()` requires parentheses in Python 3.

3.2 Python Libraries

You should not download custom libraries that do not come with Python. Of the ones that come with Python, we think you will want include `socket`, `select`, `threading`, `re`, and `time`. If you want to use another library besides the four mentioned, please ask on Piazza for approval.

3.3 Directory Structure

The directory structure of the project is as follows:

```
a3
├── tests
│   ├── message_count.out
│   ├── 12Commandments.txt
│   └── All of your custom tests should go in this folder as *.input files
├── outputs
│   └── Outputs from your tests should go in here
├── grading.py
├── master.py
├── constants.py
├── server.py
└── client.py
```

Assignment Project Exam Help

The files `grading.py`, `master.py`, and `constants.py` are given to you. Don't change them! Additionally, you don't have to touch `/tests/message_count.out` (used for grading your outputs) or `/tests/12Commandments.txt` (used to create the primary files that the clients want to back up). You will be working on `server.py` and `client.py`.

https://powcoder.com

We expect that you will have your own custom test cases, demonstrating that you have comprehensively tested your project, even though you will not be submitting them for grading. For each phase, you will only submit `server.py` and `client.py` on CMS. Therefore, we recommend using something like git tagging to keep track of your finished versions.

Add WeChat powcoder

4 Specification

4.1 Server

In this cloud backup project, the server has to do several very important tasks:

1. Accept connections from clients.
2. Only accept data from a client if the length of the data is below a set size (the buffer size).
3. Log important actions taken by the server (more on this later).
4. Successfully back up the clients' file (`BackupWriter` in `constants.py` helps you with this).

In order to help you with the server, we have provided the file `server.py` with some starter code. The first thing to notice is that `Server` extends the `threading.Thread` class. This shouldn't affect how you implement the class except that you need to implement the `run` function. We suggest looking at the Python documentation. The second thing you should notice is that `__init__` takes in three arguments. These correspond to the address (most likely "`localhost`") and port the server socket is being hosted on along with the filepath to the test being currently run. This last argument is important because the files that the server needs to generate must be named based off of the name of the `.input` file (don't worry if you don't understand this part yet).

4.2 Client

Similar to the server, the Client has one main task: given the contents of a file, ensure that everything is successfully sent and backed up on the server side.

We have provided `client.py` to help you get started. Just like the server, `Client` extends `threading.Thread` and takes in three arguments. The first is a unique integer index given to the client ranging from 0 to 9 inclusive. The other two correspond to the address (most likely "`localhost`") and port of the server socket to which this client needs to connect.

Each client will send either no messages or a single message (contents of the file being backed up) to the server. However, the client cannot terminate its connection between itself and the server until instructed.

4.3 Language

The Nimbus 10000 supports a very specific message format. Clients wishing to use this service need to follow the specified format in order to "speak the same language" as the server. Note that "<" and ">" aren't part of the message; rather, they are just there to show that this is a parameter in the command. Additionally, `<message_size>` is the number of characters in the message after the "`:`". Your solution will need to understand and communicate with the following messages:

1. $m_{connection}$ = `<message_size>:CONNECTION|<index>`
2. m_{ready} = `<message_size>:READY`
3. m_{data} = `<message_size>:DATA|<data>`
4. $m_{response}$ = `<message_size>:RESPONSE|<status>`

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

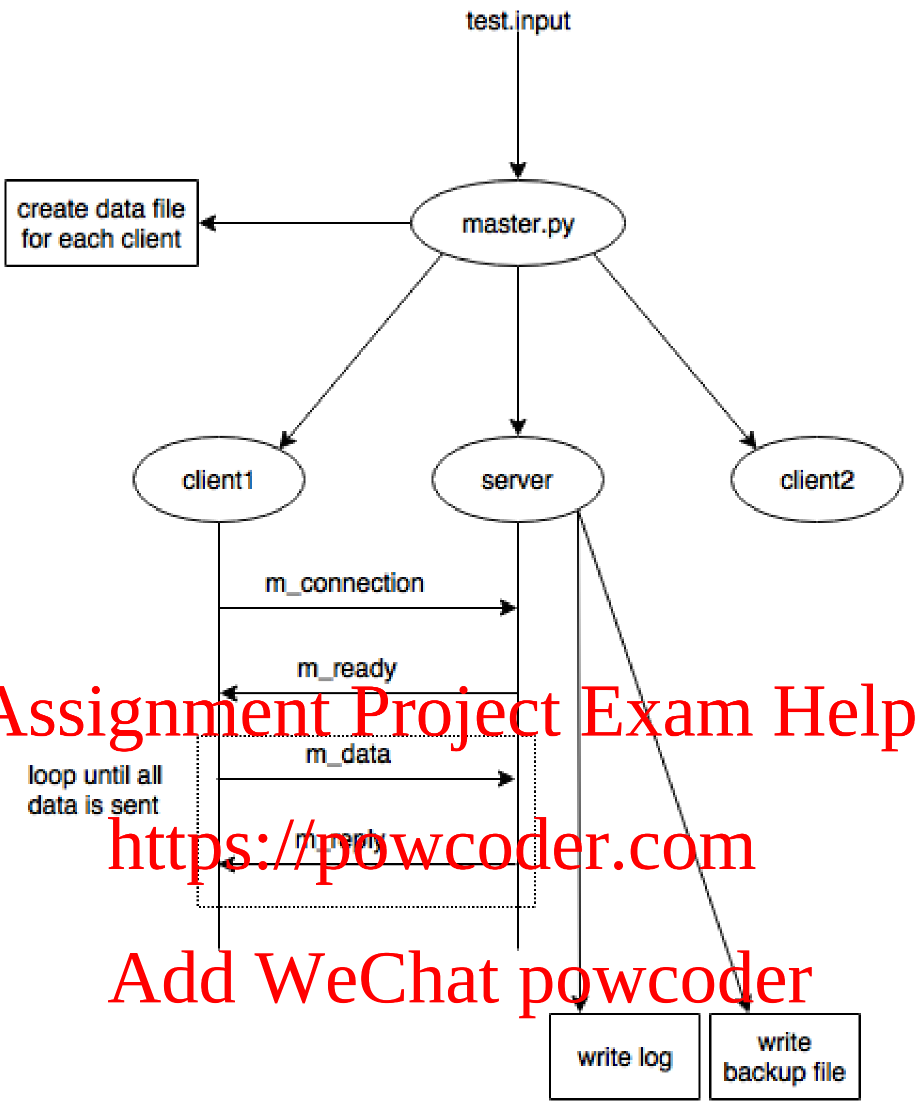
Figure 1 is a high level architecture diagram of the system.

For grading, we will compare the original data file created by the master and the backup file created by the server. We will also parse the log file created by the server to check for correctness of implementation. To ensure that there's no confusion about how to format one of these messages, we have provided functions to automatically generate the messages for you; you just need to provide the correct parameters. These functions are in `MessageGenerator` in `constants.py`. In addition to these functions, we have also provided regular expressions that you can use to help parse the messages.

4.4 Server Output

One way we will be testing your submission is based on your output file. This output file is essentially a log of the actions that the server does. To make this process easier, we have provided you `OutputWriter` in `constants.py`. To use this, you will need to initialize an instance of `OutputWriter`, after which you will be able to use the methods in the class. Remember to close the `OutputWriter`! Here are the times you will need to log something:

- Connection has started with a client (`write_connection_start`).
- Connection has ended with a client (`write_connection_end`).
- Server has received data from a client (`write_received`).
- Server has received its buffer size changed (`write_buffer_set`).



Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Figure 1: High level architecture when running a test using `master.py`

4.5 Testing API

As you have seen in class, you can run a server from one console window and run the client from another console window. Well, on the CSUG and VMs, dealing with multiple console windows is a pain. So we have made your lives easier and created `master.py`.

`master.py` is the master script that reads in an `*.input` file (each `.input` file is a separate test case) and follows the instructions outlined in the file. This file will instruct the master to do certain actions including starting up the server, the client, sending messages, setting buffer sizes, etc. `master.py` is the central Python script that links everything together and can be run via:

```
$ python3 master.py tests/<test_filename>.input
```

Making sure that `.input` files are well formatted is very important. Note that each line is a separate command (empty lines are disregarded). The following table lists out all of the valid commands to have in the `*.input` file. `master.py` will throw an exception if an invalid command is specified.

Command	Action
`start server <address> <port>`	Create an instance of Server, start running the thread, and wait half a second to ensure that the server has started. The server socket will be receiving connections on `address` (most likely "`localhost`") and port `port` (feel free to pick anything between 10000 and 29999). When writing the command, don't include the "`<`" and "`>`"; they are just to indicate that this is a parameter.
`start client <index> <address> <port>`	Create an instance of Client, start running the thread, and wait half a second to ensure that the client has started. Each instance of a client should be given a unique integer index (0-9 inclusive). The client socket will be responsible for connecting to the server with address `address` and port `port`. Do not start up the same client more than once per test.
`stop server`	Stop the server! This should cause your Server instance to finish executing, close its connections, and perform clean up. The server should be stopped after all clients that are connected to it are stopped. This command should be called exactly once on the last line of the `*.input` file.
`stop client <index>`	Stop the client with index `index`. This should cause your Client instance to finish executing, close its connection to a server, and perform clean up.
`<index> buffer <size> <num_recvs>`	Sets the server buffer size to `size` for the connection to client with index `index`. The buffer will stay at this size for `num_recvs` (the number of times the server receives data from the client regardless of whether it was successful or not). `num_recvs` can be -1 which means that the buffer will remain at the specified size forever; note that any subsequent buffer size setting would then be ignored. The buffer sizes for a connection must be set before the client sends any data to the server; it can be set even before starting the client. Buffer sizes should be initialized to 0 for each connection and returned to 0 if there are no more buffer-setting commands remaining.
`<index> send <num_characters>`	The client with index of `index` has a file of length `num_characters` that it needs the server to back up. Note that the master will generate this file automatically (saved in the `outputs` folder with a `_primary` in the name) based on an arbitrary algorithm (what is used during grading might be different than what is in `master.py`) and inform the client of the contents. Currently, the random `_primary` is being generated from the first `num_characters` of a random text; we are not trying to trick you so don't worry about edge cases where you would have to back up a file larger than this. `master.py` does not block on this command.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

`wait <seconds>`	Tell the master to wait a certain number of seconds (as a float) before processing the next command.

That was long and complicated. Here's an example of a test and what each line means:

`start server localhost 20000`	This starts up the server to receive connections at `localhost` on port 20000.
`start client 4 localhost 20000`	Start up client 4 and connect to `localhost` on port 20000. The client index doesn't always have to start with 0; it just has to be a number between 0 and 9 inclusive.
`4 buffer 50 10`	Server sets buffer size for connection with client 4 to be 50 for receives 0 through 9.
`4 buffer 70 10`	Server sets buffer size for connection with client 4 to be 70 for receives 10 through 19.
`4 buffer 20 -1`	Server sets buffer size for connection with client 4 to be 20 for receives $\geq$ 20.
`2 buffer 50 20`	Server sets buffer size for connection with client 2 to be 50 for receives 0 through 19.
`2 buffer 20 -1`	Server sets buffer size for connection with client 2 to be 20 for receives $\geq$ 20.
`start client 2 localhost 20000`	Start up client 2 and connect to `localhost` on port 20000. Note that this line came after setting the buffer sizes for this connection while it was the opposite for client 4. You can do it either way and it doesn't really matter; just make sure you set the buffer sizes before any `send`s to the client.
`4 send 2000`	Client 4 wants to backup a file that has 2000 characters and needs to send the contents to the server.
`2 send 5000`	Client 2 wants to backup a file that has 5000 characters and needs to send the contents to the server.
`stop client 4`	Client 4 should be stopped now.
`stop client 2`	Client 2 should be stopped now. Note that the order could have been switched up so that client 2 is stopped before client 4. The `*.output` should be different for the two.
`stop server`	Server should be stopped now.

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

4.6 Grading

Grading will be strictly via test cases. We will provide couple test cases to help you understand how to write test cases. Passing these test cases will be a small portion of your grade. The majority of your grade will be assessed based on test cases not provided. We may change `constants.py` in non-intrusive ways such that specifications will not be broken, but certain variables and/or message generating script might be different.

We have provided `grading.py` which is an automated grading script. It will take all of the tests in the `tests` folder and run them and grade the results. It can be invoked by calling:

```
$ python3 grading.py --phase1
```

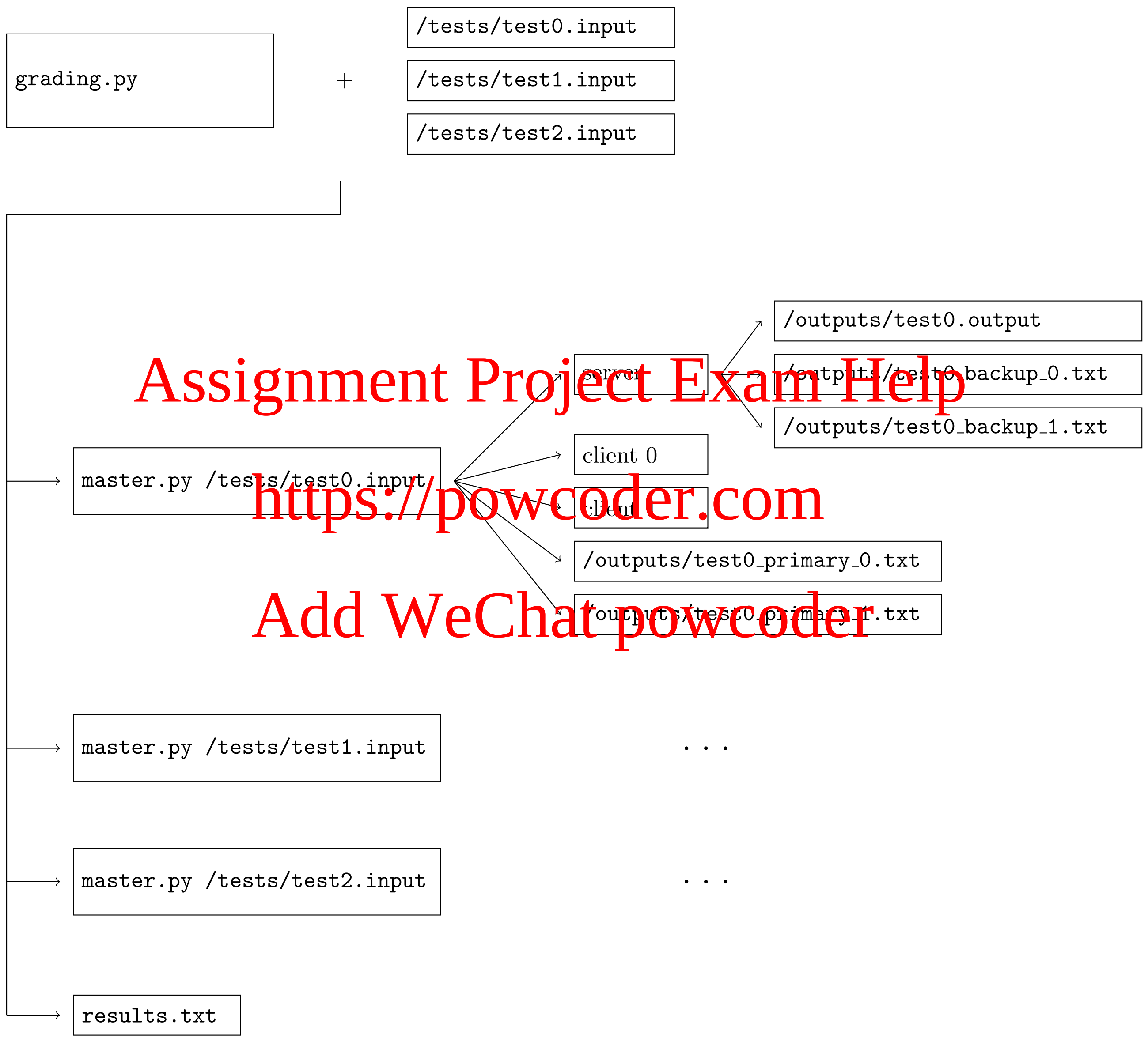
where the optional parameter at the end is only needed if you are running `grading.py` on your Phase 1 code. The results of the grading script will be in a `results.txt` file. There are 4 levels for results:

- 0: This test fully passed!
- 1: The number of messages sent in every connection doesn't match the expected number.
- 2: A specification is broken!
- 3: An exception occurred or execution of the test timed out (currently set to 10 seconds).

Different marks will be given for each level where level 0 corresponds to full points and level 3 corresponds to no points.

Intentionally fooling the test script will be considered a violation of academic integrity. For example, you cannot write a program that just creates the expected *.output and back up *.txt files.

4.7 Architecture Summary


grading.py
+
/tests/test0.input
/tests/test1.input
/tests/test2.input
/outputs/test0.output
server
/outputs/test0_backup_0.txt
/outputs/test0_backup_1.txt
master.py /tests/test0.input
client 0
client
/outputs/test0_primary_0.txt
/outputs/test0_primary_1.txt
master.py /tests/test1.input
. . .
master.py /tests/test2.input
. . .
results.txt

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Figure 2: Architectural summary

5 Phases

5.1 Phase 1 - Single Client

In this section, you will need to implement the server and client with TCP sockets starting with the provided `server.py` and `client.py` starter files.

- The server needs to handle a connection from a single client. After this client disconnects, the server should wait for the `stop` command before terminating. Clients (not the server) should be the ones proposing ending the connection with the server by closing its socket.
- The server will also have an infinite buffer size so the client can just send its entire message to the server in one m_{data}; the server should respond with a successful $m_{response}$.
- Since you are using TCP sockets, you cannot assume that the `socket.recv` function will return an entire message. You must manually check whether the message was fully received by checking the `<message_size>`. Keep reading from the socket until the complete message has been received.
- You do not need to implement `set_buffer_size` in `server.py` for Phase 1.
- When running `grading.py`, don't forget to include the option `--phase1` or your tests will fail!
- Use `RECEIVE_SIZE` variable in `constants.py` when you call `sock.recv()`.
- In Python 2, a `str` is an array of bytes while in Python 3, it is unicode string. Therefore, you must encode a string (i.e. `.encode("utf-8")`) before sending from a socket and you must decode to a string (i.e. `.decode("utf-8")`) after receiving from a socket.

Assignment Project Exam Help

Two sample test cases are provided for you for this part: `test0.input` and `test1.input`. Expected `*.output` files are also provided in the `/outputs` directory. The other tests will not work for this phase. However, note that even though Phase 2/3 should pass Phase 1 tests, the outputs will not be the same since Phase 1 doesn't have Slow Start.

https://powcoder.com

5.2 Phase 2 - AIMD

Add WeChat powcoder

- You now need to implement the `set_buffer_size` function in `server.py`. When the server receives data from a client, it must check whether or not the length of the data in m_{data} is less than the buffer size. If it is, then respond to the client with a successful $m_{response}$; otherwise, respond with a failure $m_{response}$. Remember to log all m_{data} received, regardless of whether or not it was successful or not.
- The client needs to implement TCP Slow Start when starting to send data. The suspected server buffer size starts at 1 character. After the first failed response from the server, switch to Congestion Avoidance using AIMD.
- AIMD needs to increase the suspected server buffer size by 1 for every success response and divide by 2 (floored) for every failed response. Making sure you increase/decrease the suspected server buffer size is very important because we will be testing you on how many messages (successful and failed) that were sent during the completion of the test.

An additional sample test case is provided for you for this part: `test2.input`.

5.3 Phase 3 - Multi Clients

- Now use `epoll` to handle multiple clients! After completing the first two phases of this project, you will have seen that `socket.accept` is a blocking call, i.e. that the code will be stuck on this statement until there is another incoming connection. If we have multiple clients, we have to be able to accept multiple incoming connections to the server, accept requests from these multiple clients, and respond to them all at the same time. This is where `epoll`, a Linux kernel system call for I/O event notification capable of monitoring multiple file descriptors for possible I/O events, comes in handy. It is able to simplify many threads asynchronously handling jobs into a single thread synchronously handling jobs.
- Some hints when you are using `epoll`:
  - Don't forget to set your sockets to be non-blocking.
  - You might need a timeout for your `epoll` waiting for events for your server to terminate.
  - Useful event masks include `EPOLLIN`, `EPOLLOUT`, `EPOLLHUP`.
  - Refer to the Python documentation!

An additional sample test case is provided for you for this part: `test3.input`. In this phase, it is ok for your `*.output` to not fully match the provided sample output.

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder