Assignment 2: Peer-to-Peer File Synchronizer

COMPSCI 4C03: Computer Networks

Shivom Sharma

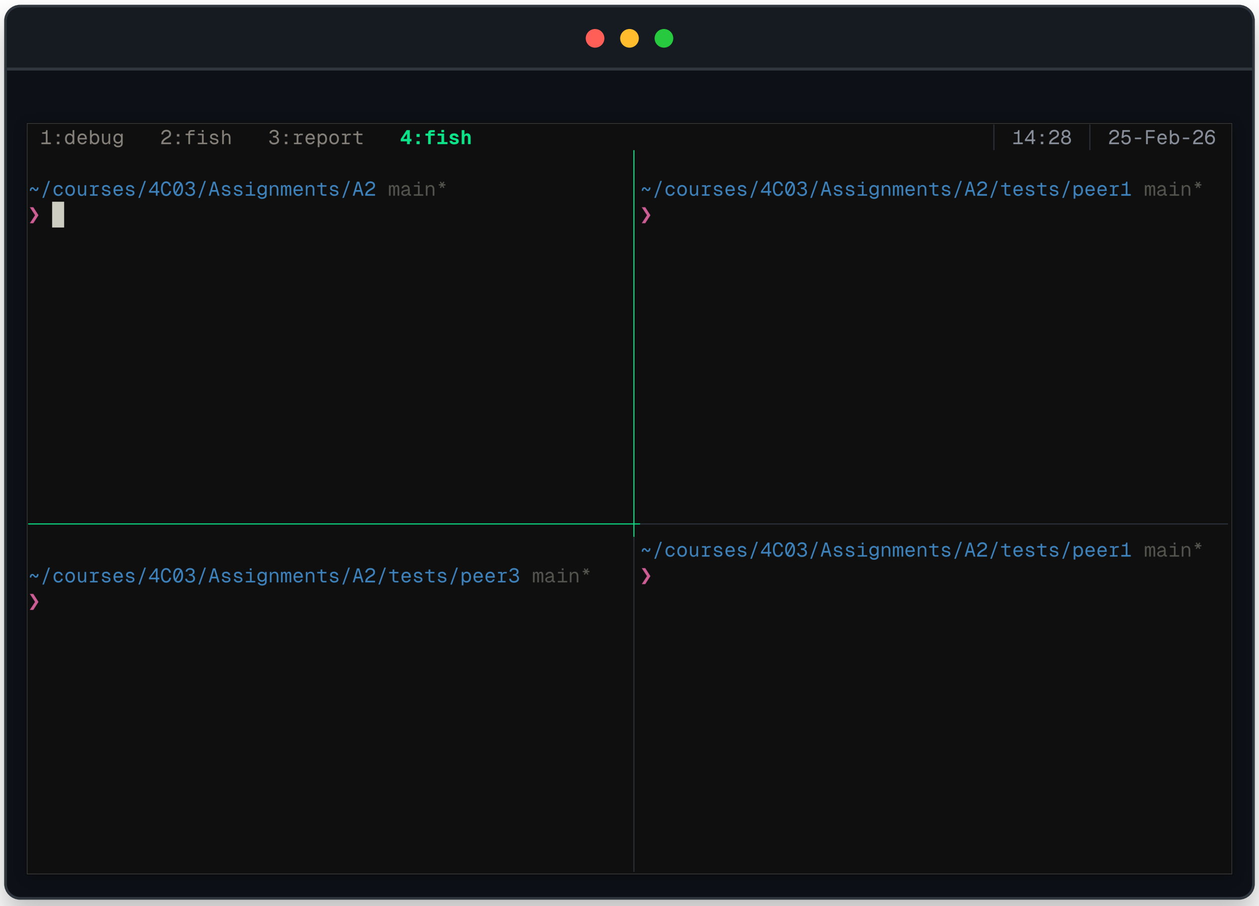
Student ID: 400332395

Instructor: Prof. Rong Zheng

Contents

1. Test Cases and Results

This document aims to summarize the methodology that I followed to design my test cases to ensure that the code was working in the proper manner end-to-end. I utilized my tmux windows to run the code with multiple peers, and also utilized the unittest framework for python to test code where it made sense to.



```
1:debug  2:fish  3:report  4:fish                                14:28  25-Feb-26

~/courses/4C03/Assignments/A2 main*
❯

~/courses/4C03/Assignments/A2/tests/peer1 main*
❯

~/courses/4C03/Assignments/A2/tests/peer3 main*
❯

~/courses/4C03/Assignments/A2/tests/peer1 main*
❯
```

Figure 1: End-to-end terminal test setup

1. Local File Info (get_file_info)

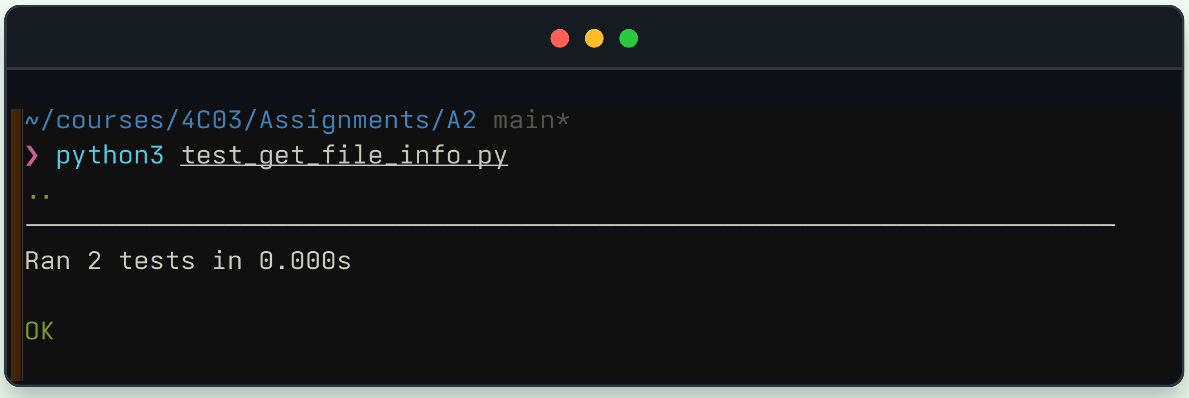
Objective: Verify that get_file_info() returns the expected local file metadata.

Test Design

I create a temporary test.txt file, then call get_file_info(), and verify that the file appears in the returned structure. The file is removed during cleanup so the test is repeatable.

```
import os.path
import unittest
from fileSynchronizer import get_file_info


class TestGetFileInfo(unittest.TestCase):
    def test_get_file_info(self):
        res = get_file_info()
        expected = []
        self.assertEqual(res, expected)

    def test_get_file_info_with_test_file(self):
        # Create test.txt
        with open("test.txt", "w") as f:
            f.write("test")

        try:
            res = get_file_info()
            # Check if test.txt is found properly
            failed = True
            for entry in res:
                if entry["name"] == "test.txt":
                    failed = False
                    break
            assert not failed
        finally:
            # Clean up: remove test.txt
            if os.path.exists("test.txt"):
                os.remove("test.txt")


if __name__ == "__main__":
    unittest.main()
```

Results: The test passes confirming the file format matches.



```
~/courses/4C03/Assignments/A2 main*
❯ python3 test_get_file_info.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Figure 2: get file info test passing

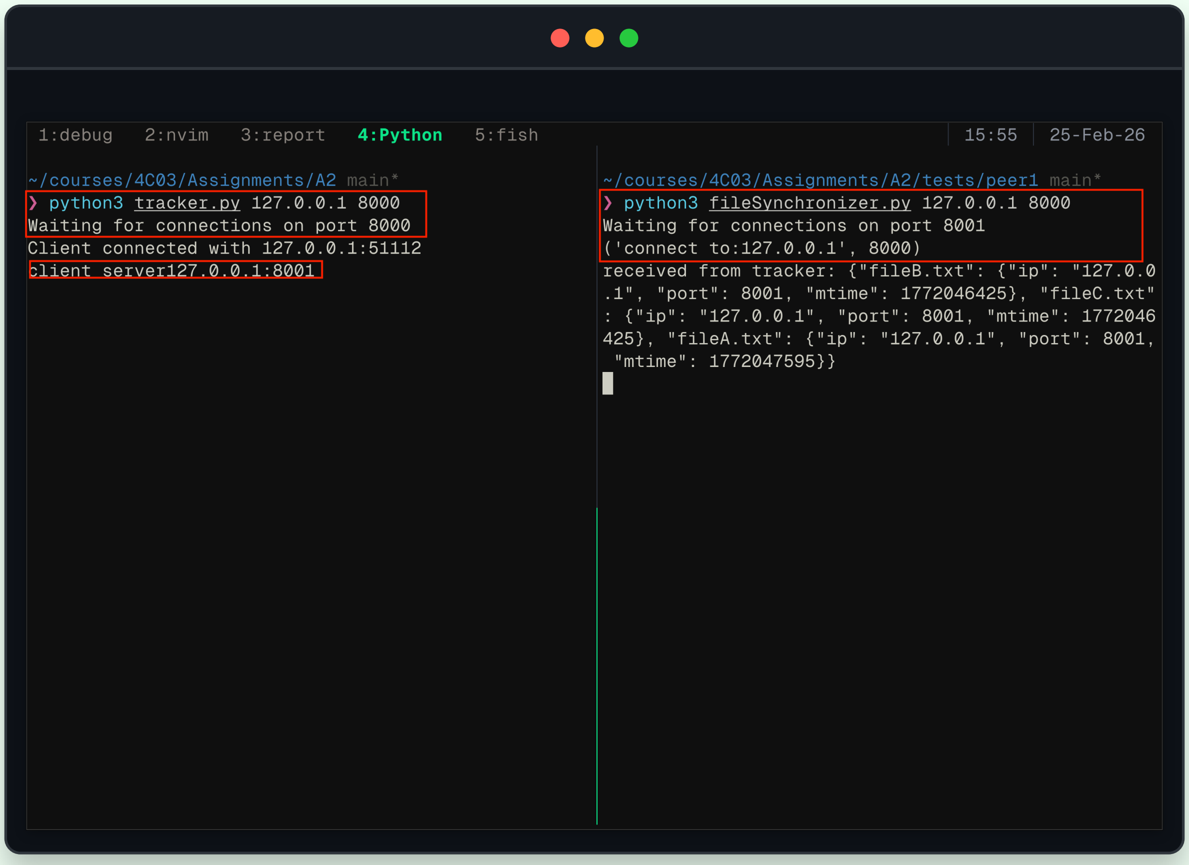
2. Next Available Port (get_next_avaliable_port)

Objective: Verify that the peer selects the next free port when preferred port is occupied.

Test Design

The tracker and peer were both started with port 8000 on the same host, to see if the next sequential port 8001 was used.

Results: We can see that the perr binds to port 8001 in the terminal screenshot.



```
1:debug  2:nvim  3:report  4:Python  5:fish                                   15:55 │ 25-Feb-26

~/courses/4C03/Assignments/A2 main*
❯ python3 tracker.py 127.0.0.1 8000
Waiting for connections on port 8000
Client connected with 127.0.0.1:51112
client server127.0.0.1:8001
```

```
~/courses/4C03/Assignments/A2/tests/peer1 main*
❯ python3 fileSynchronizer.py 127.0.0.1 8000
Waiting for connections on port 8001
('connect to:127.0.0.1', 8000)
received from tracker: {"fileB.txt": {"ip": "127.0.0
.1", "port": 8001, "mtime": 1772046425}, "fileC.txt"
: {"ip": "127.0.0.1", "port": 8001, "mtime": 1772046
425}, "fileA.txt": {"ip": "127.0.0.1", "port": 8001,
 "mtime": 1772047595}}
```

Figure 3: Peer selects next available port when initial port is occupied

3. Initialization (__init__)

Objective: Verify correct initialization of FileSynchronizer state and sockets.

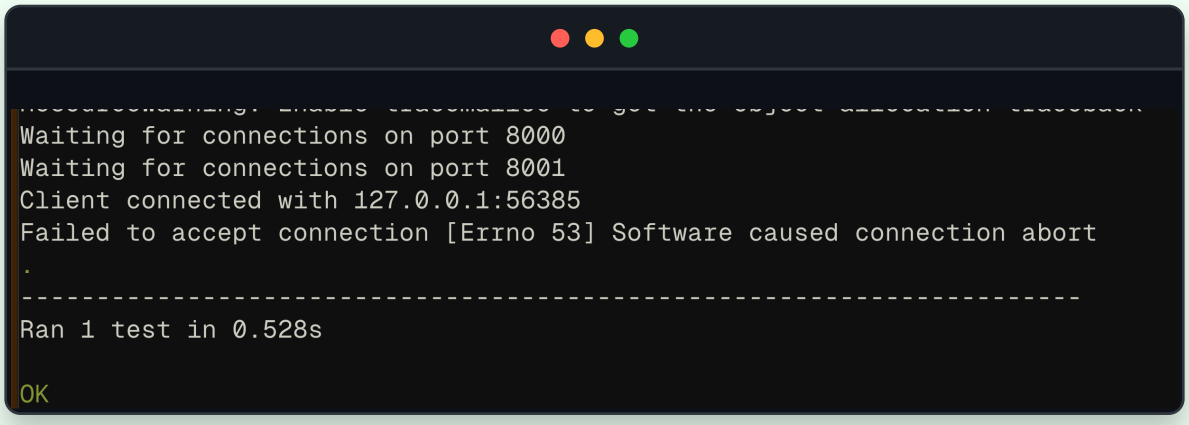
Test Design

I instantiated FileSynchronizer with known arguments and asserted key fields and socket setup values.

```python
import unittest
from fileSynchronizer import FileSynchronizer, get_next_avaliable_port
from tracker import Tracker
import time


# server socket is bound to the correct port
class testinitfilesynchronizer(unittest.TestCase):
    def test_init_file_synchronizer(self):
        tracker_host = "0.0.0.0"
        tracker_port = 8000
        tr = Tracker(tracker_port, tracker_host)
        synchronizer_port = get_next_avaliable_port(8000)
        tr.start()
        fs = FileSynchronizer(
            trackerhost=tracker_host,
            trackerport=tracker_port,
            port=synchronizer_port,
        )
        fs.start()
        time.sleep(0.5)
        sock_name = fs.server.getsockname()
        assert sock_name[1] == synchronizer_port
        # ensures that server is listening
        assert fs.server.fileno() != -1
        # check that client is properly connected

        assert fs.port == synchronizer_port
        assert fs.host == tracker_host
        assert fs.BUFFER_SIZE == 8192
        assert fs.client.timeout == 180
        expected_msg = b'{"port": 8001, "files": []}\n'
        assert fs.msg == expected_msg

        peer_name = fs.client.getpeername()
        assert peer_name[0] == "127.0.0.1"
        assert peer_name[1] == tracker_port

        fs.exit()
```

```python
if __name__ == "__main__":
    unittest.main()
```

Results: We can see the assertions passing with the test, meaning the initialization has the proper expected values listed in the test code.



```
Waiting for connections on port 8000
Waiting for connections on port 8001
Client connected with 127.0.0.1:56385
Failed to accept connection [Errno 53] Software caused connection abort
.
----------------------------------------------------------------------
Ran 1 test in 0.528s

OK
```

Figure 4: Initialization test output

4. Serving Files (process_message)

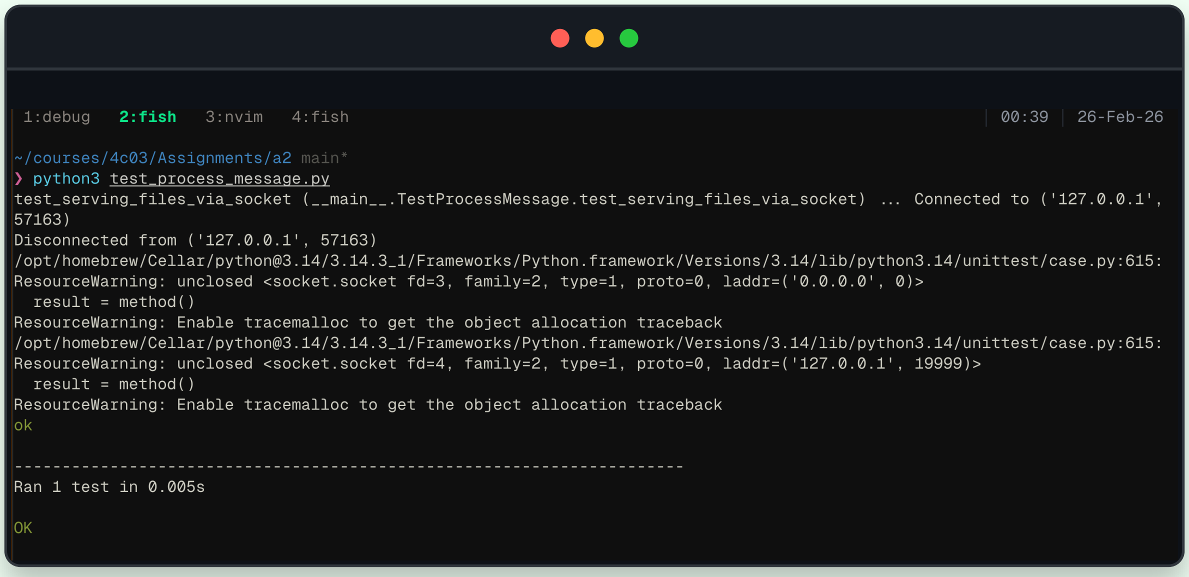
Objective: Verify that process_message() returns the requested file with the expected Content-Length header.

Test Design

I spun up a socket client and server interaction. The client sends “test.txt\n”; the server processes it via process_message(), then responds with Content-Length plus file bytes.

```python
import os
import socket
import threading
import shutil
import tempfile
import unittest

from fileSynchronizer import Buffer, FileSynchronizer


class TestProcessMessage(unittest.TestCase):
    def test_serving_files_via_socket(self):
        test_dir = tempfile.mkdtemp()
        test_file = os.path.join(test_dir, "test.txt")
        test_content = "Hello World"
        with open(test_file, "w") as f:
            f.write(test_content)

        original_dir = os.getcwd()
        os.chdir(test_dir)

        try:
            fs = FileSynchronizer("127.0.0.1", 9999, 19999, "127.0.0.1")
            fs.server.settimeout(3)

            def serve():
                conn, addr = fs.server.accept()
                fs.process_message(conn, addr)

            thread = threading.Thread(target=serve)
            thread.start()

            client = socket.socket()
            client.connect(("127.0.0.1", 19999))
            client.send(b"test.txt\n")

            buff = Buffer(client, 8192)
            header = buff.get_line()

            self.assertTrue(header.startswith("Content-Length: "))
            size = int(header.split(": ")[1])
            self.assertEqual(size, len(test_content))

            remaining = buff.buffer
            data = remaining
            while len(data) < size:
                chunk = client.recv(8192)
                if not chunk:
                    break
                data += chunk

            self.assertEqual(data.decode(), test_content)

            client.close()
            thread.join(timeout=1)

        finally:
            os.chdir(original_dir)
            shutil.rmtree(test_dir)


if __name__ == "__main__":
    unittest.main(verbosity=2)
```

Results: The client receives the correct header format and exact file content. We are able to see that it parses the exact <size> bytes.



```
 1:debug   2:fish   3:nvim   4:fish                                                    00:39   26-Feb-26

~/courses/4c03/Assignments/a2 main*
❯ python3 test_process_message.py
test_serving_files_via_socket (__main__.TestProcessMessage.test_serving_files_via_socket) ... Connected to ('127.0.0.1',
57163)
Disconnected from ('127.0.0.1', 57163)
/opt/homebrew/Cellar/python@3.14/3.14.3_1/Frameworks/Python.framework/Versions/3.14/lib/python3.14/unittest/case.py:615:
ResourceWarning: unclosed <socket.socket fd=3, family=2, type=1, proto=0, laddr=('0.0.0.0', 0)>
  result = method()
ResourceWarning: Enable tracemalloc to get the object allocation traceback
/opt/homebrew/Cellar/python@3.14/3.14.3_1/Frameworks/Python.framework/Versions/3.14/lib/python3.14/unittest/case.py:615:
ResourceWarning: unclosed <socket.socket fd=4, family=2, type=1, proto=0, laddr=('127.0.0.1', 19999)>
  result = method()
ResourceWarning: Enable tracemalloc to get the object allocation traceback
ok

----------------------------------------------------------------------
Ran 1 test in 0.005s

OK
```

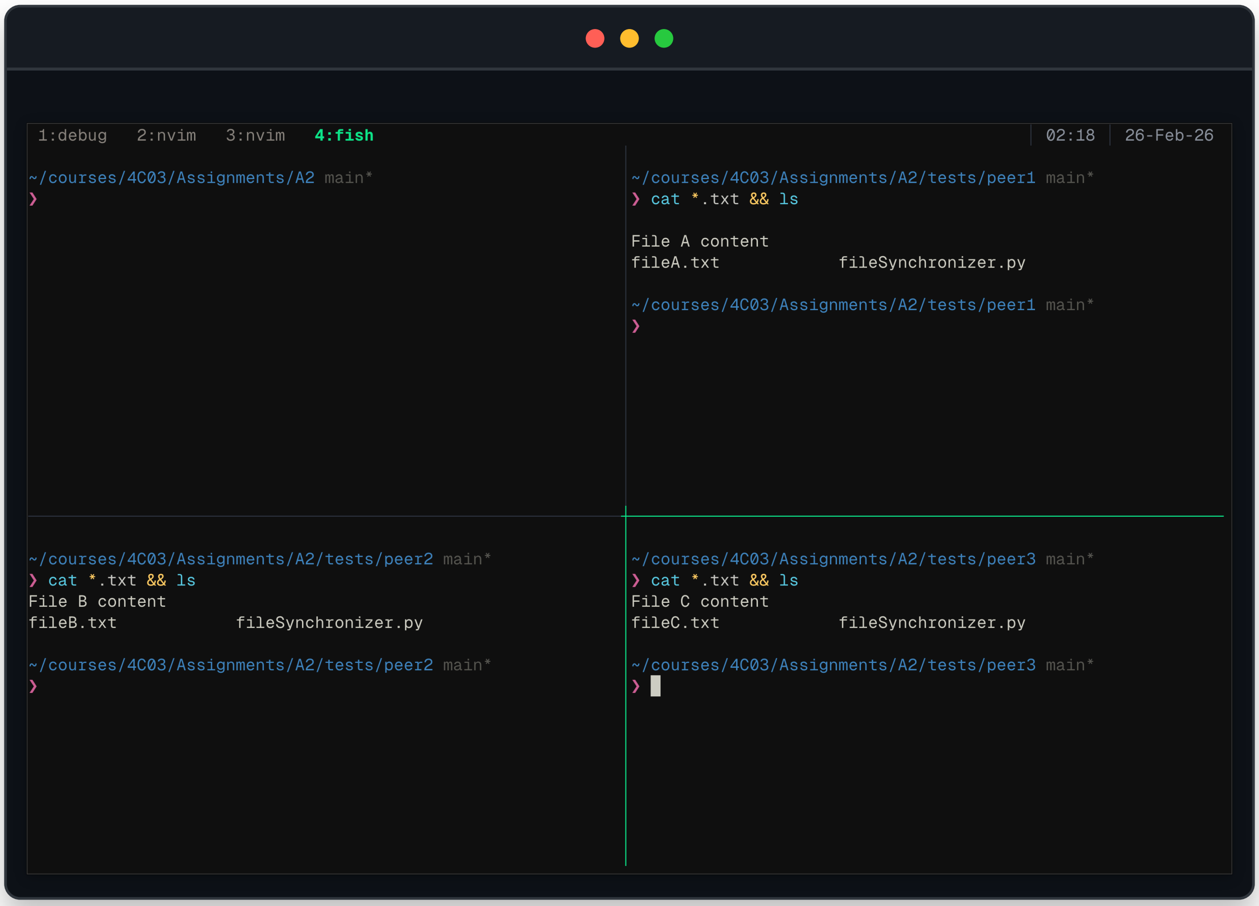
Figure 5: process_message() returns header and file payload

5. Discovering and Retrieving Files (sync)

Objective: Verify that sync() discovers missing files and retrieves them from peers.

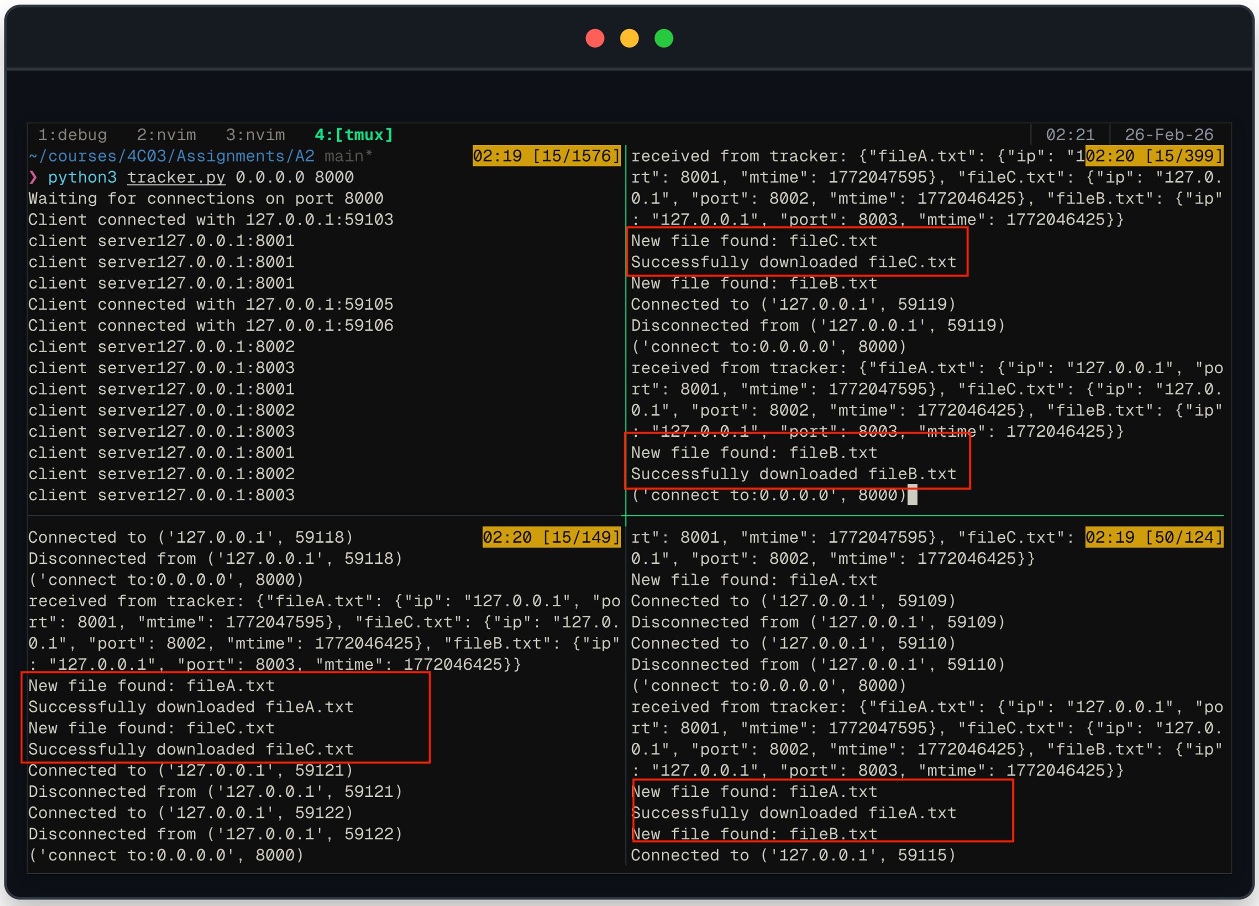
Test Design

Three peers start with different files (FileA.txt, FileB.txt, FileC.txt). After connecting with the tracker, and starting up, the peers should sync and then have the same files with all the same data. We test this using the cat command, as well as ls.



```
1:debug  2:nvim  3:nvim  4:fish                                  02:18  26-Feb-26

~/courses/4C03/Assignments/A2 main*
❯

~/courses/4C03/Assignments/A2/tests/peer1 main*
❯ cat *.txt && ls

File A content
fileA.txt            fileSynchronizer.py

~/courses/4C03/Assignments/A2/tests/peer1 main*
❯

~/courses/4C03/Assignments/A2/tests/peer2 main*
❯ cat *.txt && ls
File B content
fileB.txt            fileSynchronizer.py

~/courses/4C03/Assignments/A2/tests/peer2 main*
❯

~/courses/4C03/Assignments/A2/tests/peer3 main*
❯ cat *.txt && ls
File C content
fileC.txt            fileSynchronizer.py

~/courses/4C03/Assignments/A2/tests/peer3 main*
❯
```

Figure 6: Initial file setup for each peer



```
 1:debug   2:nvim   3:nvim   4:[tmux]                                                02:21  26-Feb-26
```

```
~/courses/4C03/Assignments/A2 main*                                 02:19 [15/1576]
❯ python3 tracker.py 0.0.0.0 8000
Waiting for connections on port 8000
Client connected with 127.0.0.1:59103
client server127.0.0.1:8001
client server127.0.0.1:8001
client server127.0.0.1:8001
Client connected with 127.0.0.1:59105
Client connected with 127.0.0.1:59106
client server127.0.0.1:8002
client server127.0.0.1:8003
client server127.0.0.1:8001
client server127.0.0.1:8002
client server127.0.0.1:8003
client server127.0.0.1:8001
client server127.0.0.1:8002
client server127.0.0.1:8003
```

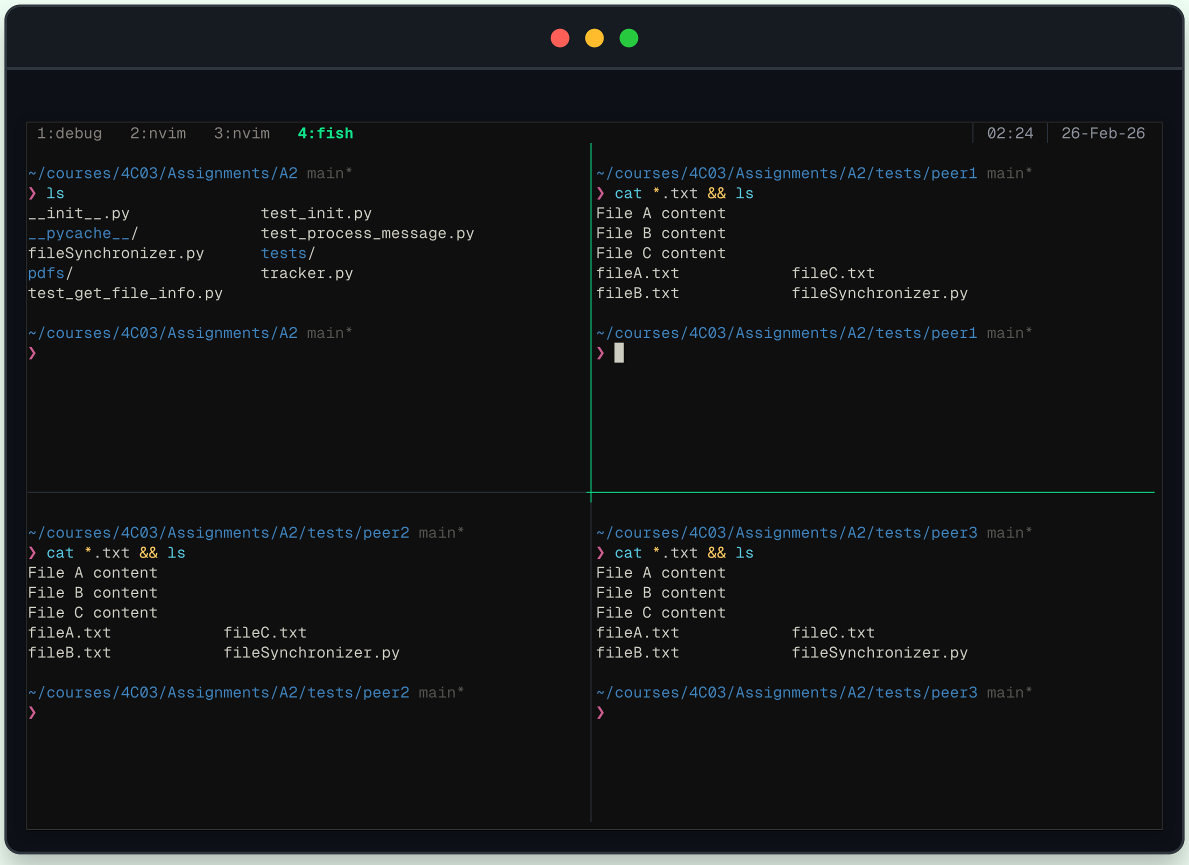
```
received from tracker: {"fileA.txt": {"ip": "1 02:20 [15/399]
rt": 8001, "mtime": 1772047595}, "fileC.txt": {"ip": "127.0.
0.1", "port": 8002, "mtime": 1772046425}, "fileB.txt": {"ip"
: "127.0.0.1", "port": 8003, "mtime": 1772046425}}
New file found: fileC.txt
Successfully downloaded fileC.txt
New file found: fileB.txt
Connected to ('127.0.0.1', 59119)
Disconnected from ('127.0.0.1', 59119)
('connect to:0.0.0.0', 8000)
received from tracker: {"fileA.txt": {"ip": "127.0.0.1", "po
rt": 8001, "mtime": 1772047595}, "fileC.txt": {"ip": "127.0.
0.1", "port": 8002, "mtime": 1772046425}, "fileB.txt": {"ip"
: "127.0.0.1", "port": 8003, "mtime": 1772046425}}
New file found: fileB.txt
Successfully downloaded fileB.txt
('connect to:0.0.0.0', 8000)
```

```
Connected to ('127.0.0.1', 59118)                                   02:20 [15/149]
Disconnected from ('127.0.0.1', 59118)
('connect to:0.0.0.0', 8000)
received from tracker: {"fileA.txt": {"ip": "127.0.0.1", "po
rt": 8001, "mtime": 1772047595}, "fileC.txt": {"ip": "127.0.
0.1", "port": 8002, "mtime": 1772046425}, "fileB.txt": {"ip"
: "127.0.0.1", "port": 8003, "mtime": 1772046425}}
New file found: fileA.txt
Successfully downloaded fileA.txt
New file found: fileC.txt
Successfully downloaded fileC.txt
Connected to ('127.0.0.1', 59121)
Disconnected from ('127.0.0.1', 59121)
Connected to ('127.0.0.1', 59122)
Disconnected from ('127.0.0.1', 59122)
('connect to:0.0.0.0', 8000)
```

```
rt": 8001, "mtime": 1772047595}, "fileC.txt": 02:19 [50/124]
0.1", "port": 8002, "mtime": 1772046425}}
New file found: fileA.txt
Connected to ('127.0.0.1', 59109)
Disconnected from ('127.0.0.1', 59109)
Connected to ('127.0.0.1', 59110)
Disconnected from ('127.0.0.1', 59110)
('connect to:0.0.0.0', 8000)
received from tracker: {"fileA.txt": {"ip": "127.0.0.1", "po
rt": 8001, "mtime": 1772047595}, "fileC.txt": {"ip": "127.0.
0.1", "port": 8002, "mtime": 1772046425}, "fileB.txt": {"ip"
: "127.0.0.1", "port": 8003, "mtime": 1772046425}}
New file found: fileA.txt
Successfully downloaded fileA.txt
New file found: fileB.txt
Connected to ('127.0.0.1', 59115)
```

Figure 7: Discovered and downloading

Results: The final directory shows that the content matches in each peer.



```
1:debug  2:nvim  3:nvim  4:fish                                            02:24 | 26-Feb-26

~/courses/4C03/Assignments/A2 main*
❯ ls
__init__.py              test_init.py
__pycache__/             test_process_message.py
fileSynchronizer.py      tests/
pdfs/                    tracker.py
test_get_file_info.py

~/courses/4C03/Assignments/A2 main*
❯
```

```
~/courses/4C03/Assignments/A2/tests/peer1 main*
❯ cat *.txt && ls
File A content
File B content
File C content
fileA.txt            fileC.txt
fileB.txt            fileSynchronizer.py

~/courses/4C03/Assignments/A2/tests/peer1 main*
❯
```

```
~/courses/4C03/Assignments/A2/tests/peer2 main*
❯ cat *.txt && ls
File A content
File B content
File C content
fileA.txt            fileC.txt
fileB.txt            fileSynchronizer.py

~/courses/4C03/Assignments/A2/tests/peer2 main*
❯
```

```
~/courses/4C03/Assignments/A2/tests/peer3 main*
❯ cat *.txt && ls
File A content
File B content
File C content
fileA.txt            fileC.txt
fileB.txt            fileSynchronizer.py

~/courses/4C03/Assignments/A2/tests/peer3 main*
❯
```

Figure 8: All content matches

6. Overwriting Outdated Files (sync)

Objective: Verify that sync() overwrites an older local copy when a newer peer version exists.

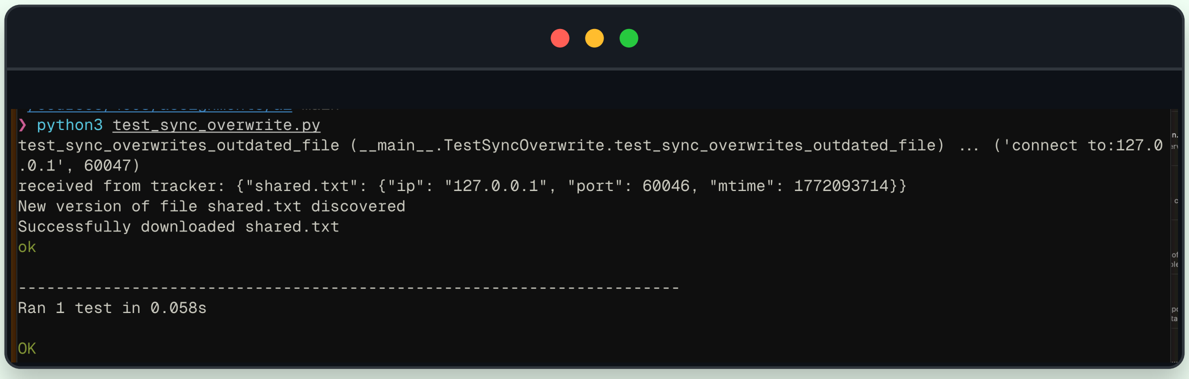
Test Design

Peer A and Peer B start with the same filename but different content and mtimes. Peer A holds the newer version. Peer B runs sync() using a tracker response that points to Peer A as the latest source.

```
import json
import os
import socket
import tempfile
import threading
import time
import unittest
from unittest.mock import patch

from fileSynchronizer import Buffer, FileSynchronizer


class NoOpTimer:
    def __init__(self, *_args, **_kwargs):
        pass

    def start(self):
        pass


class TestSyncOverwrite(unittest.TestCase):
    def test_sync_overwrites_outdated_file(self):
        with tempfile.TemporaryDirectory(prefix="sync_overwrite_") as root:
            peer_a = os.path.join(root, "peerA")
            peer_b = os.path.join(root, "peerB")
            os.makedirs(peer_a)
            os.makedirs(peer_b)

            filename = "shared.txt"
            a_file = os.path.join(peer_a, filename)
            b_file = os.path.join(peer_b, filename)

            with open(a_file, "w", encoding="utf-8") as f:
                f.write("new version from peer A")
            with open(b_file, "w", encoding="utf-8") as f:
                f.write("old version")

            old_mtime = int(time.time()) - 120
            new_mtime = old_mtime + 60
            os.utime(a_file, (new_mtime, new_mtime))
            os.utime(b_file, (old_mtime, old_mtime))

            file_peer_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            file_peer_server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
            file_peer_server.bind(("127.0.0.1", 0))
            file_peer_port = file_peer_server.getsockname()[1]
            file_peer_server.listen(1)

            def file_peer_run():
                try:
                    conn, _ = file_peer_server.accept()
                    with conn:
                        requested = Buffer(conn, 8192).get_line()
                        if requested == filename:
                            with open(a_file, "rb") as f:
                                data = f.read()
                            conn.sendall(
                                f"Content-Length: {len(data)}\n".encode("utf-8")
                            )
                            time.sleep(0.05)
                            conn.sendall(data)
                finally:
                    file_peer_server.close()

            file_peer_thread = threading.Thread(target=file_peer_run, daemon=True)
            file_peer_thread.start()

            tracker_payload = {
                filename: {
                    "ip": "127.0.0.1",
                    "port": file_peer_port,
                    "mtime": new_mtime,
                }
            }

            tracker_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            tracker_server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
            tracker_server.bind(("127.0.0.1", 0))
            tracker_port = tracker_server.getsockname()[1]
            tracker_server.listen(1)

            def tracker_run():
                try:
                    conn, _ = tracker_server.accept()
                    with conn:
                        _ = Buffer(conn, 8192).get_line()
                        conn.sendall(
                            (json.dumps(tracker_payload) + "\n").encode("utf-8")
                        )
                finally:
                    tracker_server.close()

            tracker_thread = threading.Thread(target=tracker_run, daemon=True)
            tracker_thread.start()

            fs = None
```

```
            original_cwd = os.getcwd()
            try:
                os.chdir(peer_b)
                fs = FileSynchronizer("127.0.0.1", tracker_port, 0, "127.0.0.1")
                fs.client.connect(("127.0.0.1", tracker_port))
                with patch("fileSynchronizer.threading.Timer", NoOpTimer):
                    fs.sync()
            finally:
                os.chdir(original_cwd)
                if fs is not None:
                    fs.server.close()
                    fs.client.close()
                file_peer_thread.join(timeout=2)
                tracker_thread.join(timeout=2)

            with open(b_file, "r", encoding="utf-8") as f:
                self.assertEqual(f.read(), "new version from peer A")
            self.assertEqual(int(os.path.getmtime(b_file)), new_mtime)


if __name__ == "__main__":
    unittest.main(verbosity=2)
```

Results: Peer B's file content is replaced with Peer A's content, and Peer B's file mtime is updated to the newer timestamp.



```
❯ python3 test_sync_overwrite.py
test_sync_overwrites_outdated_file (__main__.TestSyncOverwrite.test_sync_overwrites_outdated_file) ... ('connect to:127.0
.0.1', 60047)
received from tracker: {"shared.txt": {"ip": "127.0.0.1", "port": 60046, "mtime": 1772093714}}
New version of file shared.txt discovered
Successfully downloaded shared.txt
ok

----------------------------------------------------------------------
Ran 1 test in 0.058s

OK
```

Figure 9: Overwriting result with sync()

7. Failure Handling

Objective: Verify graceful behavior under tracker/peer timeout or disconnect scenarios.

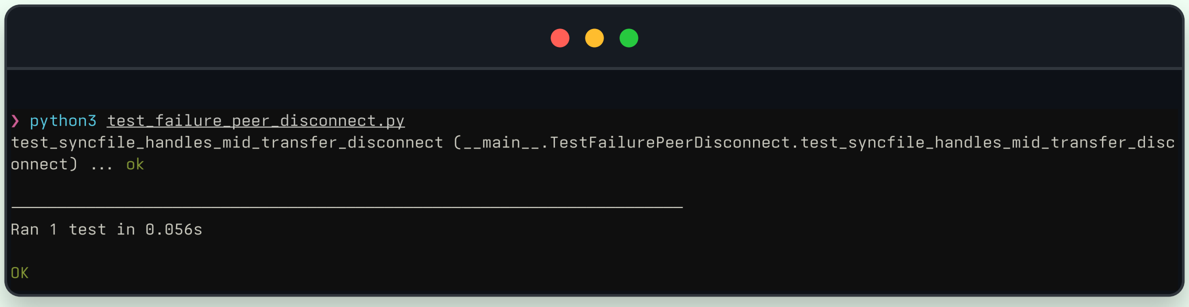
Test Design

I tested two major cases in this section, the first being a peer disconnecting during file transfer, ensuring that there were no partial files leftover and that there was a graceful exit. The second case being a check to see if tracker failure occurred properly on timeout.

```
import os
import socket
import tempfile
import threading
import time
import unittest

from fileSynchronizer import FileSynchronizer


class TestFailurePeerDisconnect(unittest.TestCase):
    def test_syncfile_handles_mid_transfer_disconnect(self):
        with tempfile.TemporaryDirectory(prefix="failure_peer_disconnect_") as
test_dir:
            original_cwd = os.getcwd()
            os.chdir(test_dir)

            fs = FileSynchronizer("127.0.0.1", 9999, 0, "127.0.0.1")

            peer_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            peer_server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
            peer_server.bind(("127.0.0.1", 0))
            peer_port = peer_server.getsockname()[1]
            peer_server.listen(1)

            filename = "broken.txt"
            advertised_size = 20
            partial_data = b"short"
            mtime = int(time.time())

            def peer_run():
                try:
                    conn, _ = peer_server.accept()
                    with conn:
                        _ = conn.recv(1024)
                        conn.sendall(
                            f"Content-Length: {advertised_size}\n".encode("utf-8")
                        )
                        time.sleep(0.05)
                        conn.sendall(partial_data)
                finally:
                    peer_server.close()

            peer_thread = threading.Thread(target=peer_run, daemon=True)
            peer_thread.start()

            try:
                fs.syncfile(
                    filename,
                    {"ip": "127.0.0.1", "port": peer_port, "mtime": mtime},
                )
            finally:
                fs.server.close()
                fs.client.close()
                peer_thread.join(timeout=2)
                os.chdir(original_cwd)

            self.assertFalse(os.path.exists(filename))
            self.assertFalse(os.path.exists(filename + ".part"))


if __name__ == "__main__":
    unittest.main(verbosity=2)
```

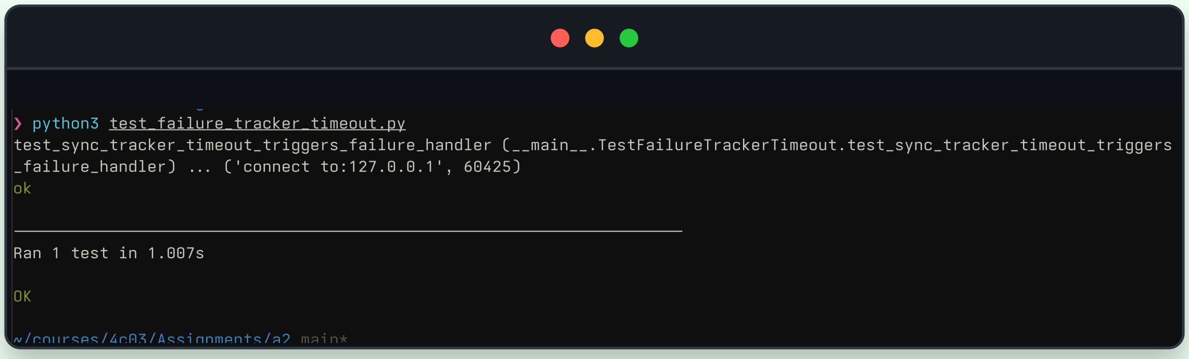
```
import socket
import threading
import time
import unittest
from unittest.mock import patch

from fileSynchronizer import FileSynchronizer


class TestFailureTrackerTimeout(unittest.TestCase):
    def test_sync_tracker_timeout_triggers_failure_handler(self):
        tracker_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        tracker_server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        tracker_server.bind(("127.0.0.1", 0))
        tracker_port = tracker_server.getsockname()[1]
        tracker_server.listen(1)

        def tracker_run():
            try:
                conn, _ = tracker_server.accept()
                with conn:
                    _ = conn.recv(8192)
                    time.sleep(1)
            finally:
                tracker_server.close()

        tracker_thread = threading.Thread(target=tracker_run, daemon=True)
        tracker_thread.start()

        fs = FileSynchronizer("127.0.0.1", tracker_port, 0, "127.0.0.1")
        fs.client.settimeout(0.2)
        try:
            fs.client.connect(("127.0.0.1", tracker_port))

            with patch.object(
                fs,
                "fatal_tracker",
                side_effect=RuntimeError("fatal_tracker_called"),
            ) as fatal:
                with self.assertRaises(RuntimeError):
                    fs.sync()

                self.assertTrue(fatal.called)
                self.assertIn(
                    "Failed waiting for the tracker to response",
                    fatal.call_args[0][0],
                )
        finally:
            fs.server.close()
            fs.client.close()
            tracker_thread.join(timeout=2)


if __name__ == "__main__":
    unittest.main(verbosity=2)
```

Results: We can see that the test passes, where we don't leave any partial traces and tracker failure properly occurs.



```
❯ python3 test_failure_peer_disconnect.py
test_syncfile_handles_mid_transfer_disconnect (__main__.TestFailurePeerDisconnect.test_syncfile_handles_mid_transfer_disc
onnect) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.056s

OK
```

Figure 10: Peer disconnect partway result



```
❯ python3 test_failure_tracker_timeout.py
test_sync_tracker_timeout_triggers_failure_handler (__main__.TestFailureTrackerTimeout.test_sync_tracker_timeout_triggers
_failure_handler) ... ('connect to:127.0.0.1', 60425)
ok

----------------------------------------------------------------------
Ran 1 test in 1.007s

OK

~/courses/4c03/Assignments/a2 main*
```

Figure 11: Tracker failure test result