Sonatus Coding Assignment User Guide

🎯 IEEE 830-1998 국제 표준에 따른 소프트웨어 개발 단계 명세 😛

💡 1. Project Overview (프로젝트 개요)

This project involves implementing a **Python-based server and client** that communicate over **TCP**. The client sends sequential steps to the server, each with specified timeout and interval settings, while the server verifies the order and returns responses with error codes as needed.

이 프로젝트는 **TCP**를 통해 통신하는 **Python 기반의 서버와 클라이언트**를 구현하는 것입니다. 클라이언트는 각 단계에 지정된 타임아웃과 대기 시간 설정에 따라 순차적으로 서버에 데이터를 전송하며, 서버는 순서를 검증하고 필요한 경우 에러 코드를 포함하여 응답합니다.

💡 2. Requirement Analysis (요구사항 분석)

아래 요구사항을 만족시키는 Server/Client 코드를 Python으로 구현해주세요.

1. Python 으로 작성 (**client.py, server.py**)
2. Server와 Client는 TCP 로 통신할 수 있도록 구현. (IP와 Port는 임의로 할당)
3. Client는 json 파일을 로드한다. json 파일은 10 단계의 Step으로 구성되어 있으며, 각 step은 timeout, 그리고 interval(다음 step id 를 보내기 까지 지연시간)을 속성으로 가진다.
   a.
   step_id : step id (유효 범위: 1~10)
   b.
   timeout : client가 server로부터 응답을 기다리는 시간 (단위: 초)
   c.
   interval : client가 server로부터 응답을 받은 후, 다음 step으로 진행하기까지 대기 시간 (단위: 초)

(json file 예시)

```
{
  "test_services": [
    {
      "step_id": 1,
      "timeout": 2,
      "interval": 10
    },
    ......
  ]
}
```

4. Client는 Server에게 step id를 데이터로 전송한다.
5. Server는 step id가 순차적으로 증가하는 지를 체크해야 하며, **step id** 와 **error code**를 데이터로 client에게 응답한다.

(ErrorCode 예시)

```
class ErrorCode(Enum):
    OK = 0
    NOK = 1
```

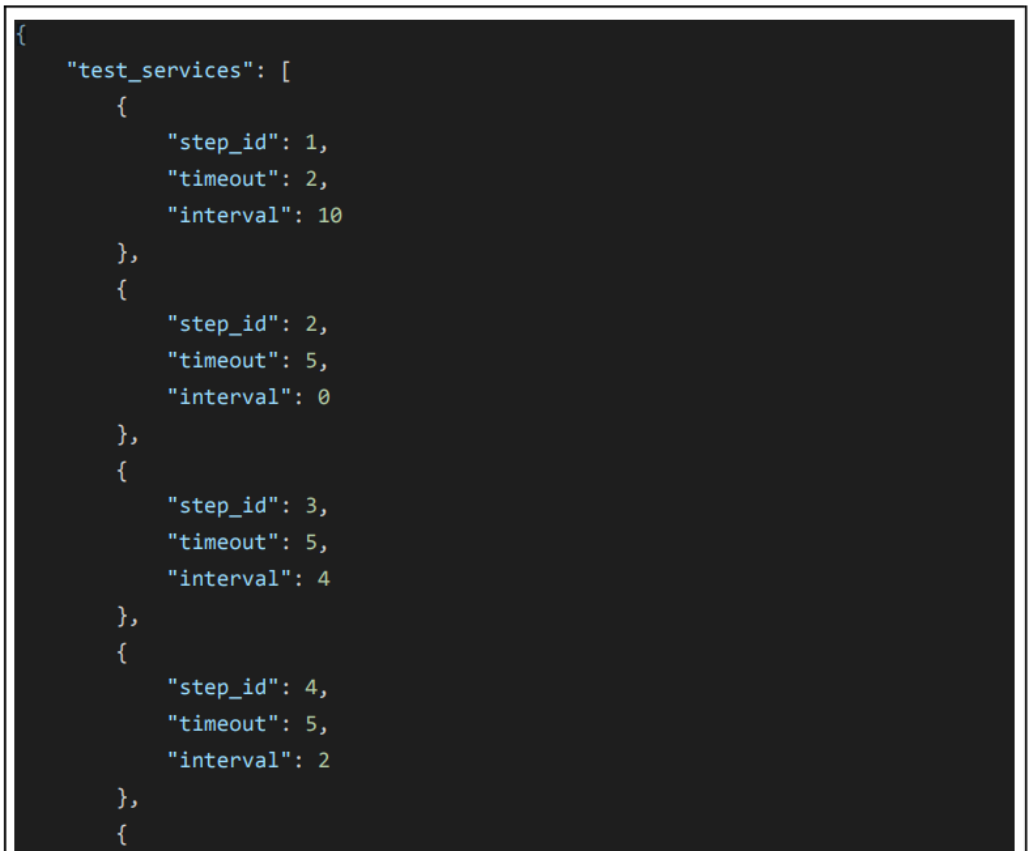
6. Client는 timeout 내에 Server로 부터 응답이 오는 지 체크한다. timeout이 발생 하거나 error code 를 수신 받을 경우, 실패 이유와 step id를 로깅 하고 다음 스텝을 진행한다.
7. 한 개의 스텝이 완료된 후에는 interval 시간 만큼 대기 후 다음 스텝을 진행한다.
8. **pcap과 설명자료를 테스트 결과로 제출한다.**

(보너스 점수)

● 서버가 다수의 클라이언트를 지원할 수 있도록 구현

● Timeout을 발생시키는 테스트 케이스와 테스트 수행 결과 첨부

Test json file:



```
{
    "test_services": [
        {
            "step_id": 1,
            "timeout": 2,
            "interval": 10
        },
        {
            "step_id": 2,
            "timeout": 5,
            "interval": 0
        },
        {
            "step_id": 3,
            "timeout": 5,
            "interval": 4
        },
        {
            "step_id": 4,
            "timeout": 5,
            "interval": 2
        },
        {
```

3. Software Architecture (소프트웨어 설계)

아래와 같은 구조로서 **Sonatus Assignment**라는 **repo** 아래에, 각각 기능을 분리해서 별도의 디렉토리로 파일을 관리합니다.

docs 폴더에는 개발자와 비개발자를 위한 문서화된 text document가 위치하고,

pcap 폴더에는 성공한 테스트 케이스와 실패한 테스트 케이스의 **packet capture** 파일이 위치하며,

src 폴더에는 주된 메인 로직을 담을 server.py와 client.py 파일이 위치하고,

test_data 폴더에는 성공 테스트 데이터와 실패 테스트 데이터 파일이 위치하며,

requirements.txt 파일은 루트폴더에 이 프로그램 실행에 필요한 library들의 버전정보가 위치합니다.

(추후 배포할 필요시에, LICENSE 파일도 포함할 수 있습니다)

```
SONATUS_ASSIGNMENT/
├── docs/
│   ├── overview.pdf
│   └── README.md
```

```
├── pcap/
│   └── success_capture.pcap
│   └── failure_capture.pcap
├── src/
│   ├── client.py
│   └── server.py
├── test_data/
│   ├── failure_data.json
│   └── success_data.json
└── requirements.txt
└── (LICENSE.md)
```

💡 4. Implementation (구현)

- **Bird-eye View (조감도)**

  우선, test_data의 success_data.json와 failure_data.json 데이터의 구조부터 살펴보았습니다.

  프로젝트에서 규정하고 있는 step_id는 1부터 10까지 순차적으로 increment 되어야 하며, client가 server로부터 응답을 받기 위한 timeout의 값과 client가 server로 부터 응답을 받은 후 다음 step을 진행하기까지 대기하는 interval이 3대 요소입니다.

  클라이언트는 서버에 데이처 처리 및 반환 요청을 하는 입장이며, 서버는 그 처리와 반환을 담당하는 기본 전제를 따라서, server.py을 우선적으로 설계했습니다.

- **server.py (서버)**

  문제에서 주어진 (보너스) edge case들 중에서, 서버가 다수의 클라이언트를 지원하도록 하려면, 멀티쓰레딩 library인 threading을 사용해야 합니다. 또한, timeout 역치를 사용자가 컨트롤 할 수 있게 하려면, 처음 실행 때 input으로 주입받을 수 있습니다.

  위와 같은 이유로, class을 instantiate 할 때, argparse 라이브러리를 통해서, 사용자로부터 각각 host, port 그리고 timeout 정보를 입력 받습니다. (만약에, 주입받지 않는다면, default 값을 설정합니다)

  멀티쓰레딩의 일환으로, def run이 함수가 아닌, def start 함수로 구현하였으며, 초기 서버 소켓과 클라이언트 소켓과의 연결성을 확보합니다. 또한, 다중 클라이언트의 동시 접속방지로 인한 데이터의 무결성과 일관성 방지를 위해서 threading안의 lock을 사용합니다.

클라이언트 소켓과 통신을 할때에, json으로 데이터를 주고 받으며, 미리 정의해둔 에러 Enum class에 맞게 예외처리를 하면서 해당 응답들을 클라이언트에게 넘겨줍니다.

- **client.py (클라이언트)**

  클라이언트 파일에서 최초로 해야하는 부분은, 성공과 실패 json 파일중에서 어떤 파일을 읽을 지 flag, 즉, argument로 설정해주는 것입니다. 또한, 서버 파일과 마찬가지고, host과 port을 사용자로 부터 주입받습니다.

  클라이언트 파일에서는 서버에게 step_id과 timeout 정보를 넘겨주며, 다음 step을 위한 interval만큼 쉬면서 처리를 해줍니다.

  여기서 핵심적인 부분은 클라이언트 소켓은 처리가 시작하고 끝날 때 마다 열고 닫혀서, 서버로부터 END 신호가 오면 루프를 종료하고 결과를 로깅합니다.

💡 5. Command Execution Method (실행 명령어 조합)

각각 server.py와 client.py를 실행하기 전에, 다음의 폴더 구조로 이동합니다. C 드라이브가 root directory라고 가정했을 때, 해당 repo인 sonatus_assignment 아래의 **src** 디렉토리까지 이동합니다.

```
C:\sonatus_assignment\src
```

그리고 우선적으로는 server.py를 실행하고, client.py를 실행하는데, argument(flag)의 조합들을 고려할 때, 아래와 같은 경우의 수가 있습니다.

Server.py 예제

현재는 디폴트 상태로서, 아래의 값들이 설정되어 있지만, 얼마든지 customize 할 수 있습니다.

- **localhost에**
- **8080 포트에**
- **테스트 파일은 success_data.json으로**

- `localhost`에서 `8080` 포트로 서버를 실행하며 타임아웃 임계값을 10초로 설정:

```
python server.py --host localhost --port 8080 --timeout 10
```

- 특정 IP(`192.168.1.100`)와 포트(`9090`)에서 타임아웃 임계값을 `7초`로 설정:

```
python server.py --host 192.168.1.100 --port 9090 --timeout 7
```

Client.py 예제

현재는 디폴트 상태로서, 아래의 값들이 설정되어 있지만, 얼마든지 customize 할 수 있습니다.

- **localhost에**
- **8080 포트에**
- **timeout은 5로**

- `localhost`의 `8080` 포트를 통해 `failure_data.json` 파일을 사용하여 클라이언트를 실행:

```
python client.py --data failure --host localhost --port 80
```

- `success_data.json` 파일을 사용하여 `192.168.1.100` IP와 `9090` 포트로 서버에 연결:

```
python client.py --data success --host 192.168.1.100 --port 9090
```

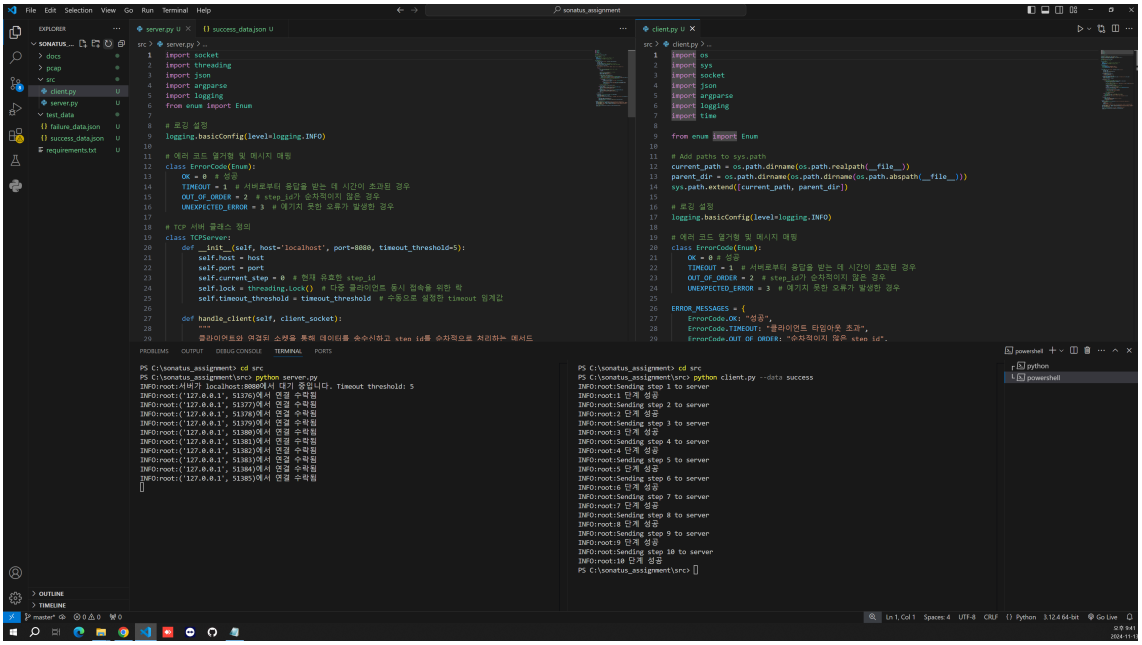
💡 6. Test & Maintenance (테스트와 유지보수)

우선적으로 success_data에서는

- **모든 step_id가 1부터 10까지 sequential하며**
- **모든 timeout이 5 이상입니다.**

즉, 만약에, step_id가 순차적으로 1부터 10으로 increment하지 않거나, timeout이 5미만이라면, success_data에서는 에러가 발생될 것 입니다.

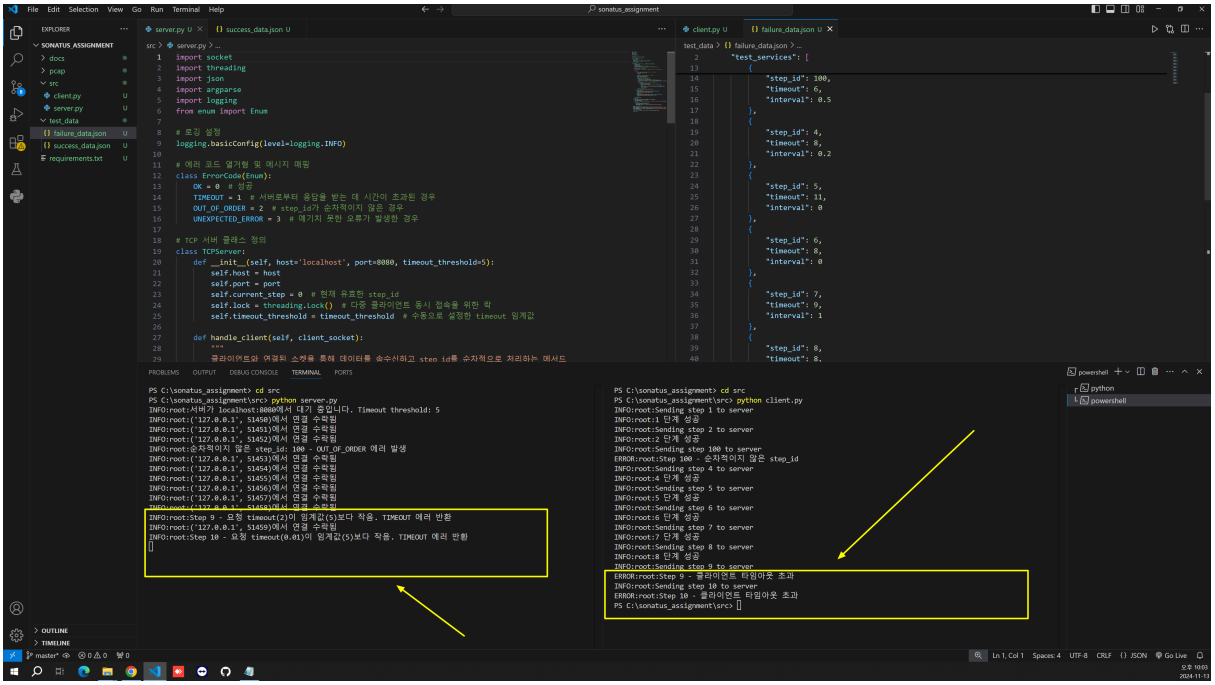
아래는 정상적으로 실행된 결과를 보여주는 스크린샷입니다.



이후의 failure_data에서는

- **step_id가 1부터 10까지 sequential하지 않으며,**
- **모든 timeout이 5 이상이지 않습니다.**

즉, 계중에 step_id가 sequential하지 않거나, timeout이 5 미만인 아이들은 실패의 결과가 아래와 같이 나오게 됩니다.



💡 7. Conclusion (프로젝트를 마치며)

Through this project, I gained a deeper understanding of building a server-client structure using **Python's TCP communication** and the importance of handling sequential data processing and error handling. The experience of using thread locks to maintain data consistency in a multi-client environment was valuable in strengthening my fundamentals in multithreading programming. As a future improvement, I would like to explore implementing asynchronous I/O to handle client requests more efficiently. Additionally, integrating the test data, currently stored in JSON files, with a database to create more r**eal-time test scenarios could be beneficial.**

이번 프로젝트를 통해 **Python의 TCP 통신**을 활용하여 서버-클라이언트 구조를 구축하고, 순차적인 데이터 처리 및 에러 핸들링의 중요성을 깊이 이해하게 되었습니다. 여러 클라이언트가 동시에 접속하는 환경에서 데이터 일관성을 유지하기 위해 **스레드 락**을 사용한 경험은 멀티스레딩 프로그래밍의 기초를 다지는 데 큰 도움이 되었습니다. 추후 개선 사항으로는 서버가 각 클라이언트의 요청을 더 효과적으로 처리할 수 있도록 **비동기 I/O**와 같은 기술을 도입해보고 싶습니다. 또한, 현재 JSON 파일로 구성된 **테스트 데이터를 데이터베이스와 연동**하여 보다 실시간적인 테스트 시나리오를 구성하는 것도 유용할 것 같습니다.