BÀI TẬP - SOCKET ĐỀ 1

Bài tập 1: Môn `CSC10008`: Mạng máy tính Khoa CNTT – CTĐA, Trường ĐH KHTN – ĐHQG TP.HCM `03/05/2020`

Bài tập cá nhân gồm một thành viên:

- `18127231`: Đoàn Đình Toàn (GitHub: @t3bol90)

Về bài tập (`Socket`)

Công việc
[1] **Server**
Crawl data kết quả xổ số về (trong ngày)
Nhận kết nối từ `client`
Nhận truy vấn từ `client`
Tìm kiếm và trả kết quả về cho `client`
[2] **Client**
Kết nối đến `server`
Gửi nội dung truy vấn đến `server`
Nhận kết quả truy vấn và hiện thị kết quả.

Tổng quan

- Mức độ hoàn thiện: Đã đạt được tất cả các yêu cầu đồ án, có thêm chức năng `crawl` kết quả về theo ngày.
- Môi trường lập trình: `JetBrains Pycharm Professional Edition` trên `Ubuntu 18.04.3 LTS`

Giao thức kết nối:

Vì giao tiếp trong quá trình dò xố số cần sự chính xác cao, sai một `bit` cũng có thể trả ra kết quả sai. Một ý nữa là trong quá trình dò xổ số, người dùng có thể chờ đợi trong một khoảng thời gian cho phép, như vậy độ trễ không phải là vấn đề trong tình huống này. Chính vì vậy em chọn `TCP` làm giao thức kết nối trong bài tập.

Mô hình bài làm:

Mô hình bài làm của em sử dụng mô hình đồng thời (`Concurrency`) và xử lí bất đồng bộ (`Asynchronous`) với thư viện `asyncio`.

Lý do chọn cách xử lí này như phân tích ở trên, bài toán ở đây là bài toán `client` gửi request tới `server` và `server` lấy kết quả từ file lên, xử lí truy vấn và trả về cho `client`. Nếu chọn hướng xử lí đồng bộ `Synchronous` thì khi 2 `clients` gửi request tới `server`, `server` chỉ có thể phục vụ 1 `client` trong 1 thời điểm nhất định và `client` còn lại vẫn phải chờ. Có nhiều cách để xử lí vấn đề này được đã được giới thiệu từ **thầy Quân** như:

- `multithreading`, sử dụng nhiều `thread` và mỗi `thread` sẽ được dành để xử lí riêng cho một `client`. Với cách xử lí như vậy, một cách gián tiếp ta đang mượn `scheduler` của `OS` để quyết định `thread` nào sẽ được chạy tiếp theo. Với ngôn ngữ `Python`, `GIL` - Global interpreter lock cũng là một vấn đề khá khó chịu khi xử lí bằng `thread`. Vì trong một thời điểm, chỉ có một `thread` giữ `GIL` được chạy và sẽ nhả `GIL` ra khi `IOwait`.
- `multiprocessing`, tạo ra nhiều `process` và cũng như `thread`, mỗi `process` sẽ xử lí cho một `client`. Cách này khắc phục được vấn đề `GIL` khi xử dụng `thread`, ta có thể chạy trên `multi-core processor`. Một vấn đề khác sinh ra là khi bạn `fork()` hay tạo ra quá nhiều `process` thì chi phí về tài nguyên là rất lớn và sẽ tăng dần, hướng đi này rất khó để phát triển ra qui mô lớn.

Qua quá trình tìm hiểu, đọc tài liệu thì em nhận ra là có một giải pháp mà em thấy khá là thú vị, mặc dù vẫn có những khuyết điểm nhất định nhưng vẫn là một trải nghiệm đáng để thử, đó là xử lí bất đồng bộ - `asynchronous`. Có nhiều cách để xử lí bất đồng bộ trong `Python` nhưng để cho đơn giản thì em dùng thẳng thư viện `asycio`. Cách làm này có ưu điểm và nhược điểm như sau:

- Xử lí được các vấn đề của `thread` như không phụ thuộc vào `GIL` và chỉ dùng một `thread`, một `process` duy nhất.
- Không phụ thuộc vào `scheduler` của `OS`, các `task` sẽ được quản lí bằng một khái niệm của `asyncio` là `event loop`.
- Tài nguyên sử dụng ít hơn so với hướng `multiprocessing` nên có khả năng phát triển rộng hơn được (`scalability`).
- Khó khăn nhất định trước mắt là phải thay đổi cách suy nghĩ về vấn đề, vì lập trình bất đồng bộ rất khác với lập trình đồng bộ (thứ theo suốt sinh viên trong quá trình từ năm nhất đến giờ).
- `Asyncio` nói riêng hay `Asynchronous` nói chung không phải là phép tiên, nó không tăng tốc quá trình xử lí thực sự của công việc mà nó chỉ là tối ưu lại thời gian và thứ tự thực hiện giữa các công việc với nhau (`cooperative multitasking`). Chính vì vậy có thể có trường hợp `client` gửi request tới `server` sẽ nhận kết quả lâu hơn so với lí thuyết (Mặc dù vẫn được phản hồi tức thì và được xử lí cùng lúc với các `client` khác).

Như vậy hướng xử lí của em không phải là hoàn hảo hay tối ưu, vẫn còn rất nhiều hướng xử lí khác và nhận được kết quả tương đương (xử lí nhiều requests cùng lúc, ít tốn tài nguyên,...) hoặc tốt hơn. Hơn hết cả thì bài tập này là một bài tập thực hành cơ bản mà thời gian nghỉ dịch thì dài nên em muốn thử nghiệm `asynchronous`.

Các nguồn tham khảo

- Using Asyncio in Python: Understanding Python's Asynchronous Programming Featuress - by Caleb Hattingh
- Python Parallel Programming Cookbook by Giancarlo Zaccone
- Medium/intro-2-asyncio
- Miguel Grinberg Asynchronous Python for the Complete Beginner PyCon 2017
- realpython/Asycio

- Devto/asyncio
- Get to grips with asyncio in Python 3 - Robert Smallshire
- Về phần lấy kết quả xố số về, em có tham khảo cách làm của bài viết: Hocpython/Scrapy

---

Các hàm chính

Ta có cấu trúc của `source code` được tổ chức như sau:

```
├── **Crawlers**
│   ├── **XSMN**
│   │   ├── **XSMN**
│   │   │   ├── items.py
│   │   │   ├── middlewares.py
│   │   │   ├── pipelines.py
│   │   │   ├── settings.py
│   │   │   └── **spiders**
│   │   │       ├── xsmt.py
│   │   │       ├── xsmb.py
│   │   │       └── xsmn.py
│   └── xs_database.db
├── **Data**
│   ├── xsmb.json
│   ├── xsmn.json
│   └── xsmt.json
├── client.py
├── server.py
└── utils.py
```

- Module `Crawlers` dùng để kéo `data` về đổ vào `xs_database.db` - một `SQL database` - rồi sau đó đẩy vào các file `xs_<mien>.json`. (Đáng lẽ bước này phải xử lí `SQL query` trực tiếp trên cái `database` nhưng mà em thấy học `SQL` trong hai tuần không nổi nên em chuyển sang xử lí file `.json`)
- `server.py` - file chứa các hàm, chương trình chạy `server` lên.
- `client.py` - file chương trình của `client`.
- `utils.py` - file chứa các hàm, `data`, kiểu dữ liệu hỗ trợ cho `server.py`.

Có thể kể các hàm quan trọng như sau:

Trong `client.py`

```
class STATE(Enum): # Kiểu dữ liệu biểu thị trạng thái của gói tin trả về từ server.
```

```
def parse_packet(data -> bytes) -> None: # Hàm phân tích gói tin từ server, hiển thị ra thông tin gói tin nhận được.
```

Trong `server.py`

```
def update_data() -> None: # Kéo data về và lưu xuống file.
def clean_stuff() -> None: # Dọn dẹp sau khi kéo data về.
```

```
async def show_tasks(): # Hiển thị thông tin của quá trình serve, quá trình xử lí và cả quá trình crawl data về. Sử dụng thư viện logging để ghi log.
```

```
def parse(data -> bytes) -> STATE,list[str]: # Phân tích gói tin gửi từ phía client, sau đó trả về trạng thái gói tin và thông tin query cần xử lí.
```

```
def client_connected_cb(client_reader, client_writer): # Hàm xử lí khi có kết nối từ phía client, đây giống như hàm main chính để gọi hàm xử lí truy vấn cho client. Xong việc thì ta sẽ gọi hàm:
def client_cleanup(fu):
# để dọn dẹp các thông tin của client vừa thoát.
async def client_task(reader, writer): # Hàm xử lí truy vấn của client.
```

```
async def main(host, port): # Hàm main giữ nhiệm vụ main loop, gọi hàm crawl data, load data lên, bind server lên và gọi các hàm handler tương ứng khi có sự kiện xảy ra (client kết nối, client ngắt kết nối).
```

Trong `utils.py`

```
class bucket_list: # Class dùng để lưu trữ xổ số của ba miền trong 3 dictionary khác nhau.
```

```
def read_data(): # Đọc data lên bucket_list
def query_handler(state, province="", pot='') -> (STATE, dict): # Hàm xử lí truy vấn từ server -> database, trả về kết quả để server gửi lại cho client.
```

Kịch bản chương trình

Khởi động `server` lên trước, `server` sẽ thực hiện những hành động sau:

- Crawl data xổ số về từ trang "http://xskt.com.vn/ket-qua-xo-so-theo-ngay". Lưu về ở dạng `.json` cho cả ba miền, sau đó load những file .json này lên để trả về cho các truy vấn của `client`.
- Treo `server` lên ở `host, port` đã định sẵn (mặc định thì `port = 5000` và `host = "127.0.0.1"`, nếu có thay đổi gì vui lòng sửa lại ở đầu file của `client.py` và `server.py`).
- Chờ request từ `client` gửi tới.
- Khi request gửi tới, `server` tiến hành:
  - Phân tích (`parse`) cú pháp của `client`, từ đó truy xuất thông tin tương ứng từ `server`:
    - Gói tin gửi đi được định nghĩa là [`STATE` (1 `byte`)] + [`MESSAGE`] với `STATE` là trạng thái trả về ứng với các tình huống còn `MESSAGE` là một `dictionary` hoặc `list` tùy theo truy vấn.
    - Nếu cú pháp là "h" hoặc "--help": Gửi lại gói tin hướng dẫn cách sử dụng `client` và các tỉnh có mở xổ số trong ngày.

        - Nếu cú pháp là "" thì `server` tiến hành truy vấn thông tin trong `data`, nếu tồn tại tỉnh trong `data` thì sẽ trả về kết quả của tỉnh đó, ngược lại trả về gói tin thông báo lỗi "Tỉnh này không có kết quả xổ số trong ngày hôm nay hoặc sai cú pháp".
        - Nếu cú pháp là " <Số cần dò>" thì `server` cũng tiến hành truy vấn thông tin trong `data`, sau đó gửi về thông báo trúng (kèm giải thưởng) hoặc không trúng.
        - Nếu sai cú pháp, `server` trả về gói tin lỗi sai cú pháp.
    - Trong tình huống `client` đóng kết nối với `server`, `server` sẽ tự dọn dẹp các thông tinh của `client` và đóng kết nối.

Về phía của `client` khi khởi động lên, `client` sẽ gửi kết nối đến `server` và chờ phản hồi (bằng gói tin `welcome`). Sau đó `client` mời người dùng nhập truy vấn rồi gửi qua cho `server`. Nếu người dùng thoát, `client` sẽ gửi truy vấn thoát (`quit`) cho `server` rồi đóng `socket` lại. `Client` sử dụng `port` bất kì còn trống trên máy của người dùng.

Cách chạy chương trình

Cài đặt các thư viện và khởi tạo môi trường ảo:

Ở đây em xài `conda`, nếu chưa có `anaconda` vui lòng cài đặt riêng.

```
conda create -n <tên môi trường> python==3.7 anaconda
```

Sau đó `activate` môi trường ảo:

```
conda activate <tên môi trường>
```

```
cd dist
```

```
pip install -r requirement.txt
```

```
cd ../src
```

Ở thư mục `src/`, dùng lệnh sau để khởi động `server`:

```
python server.py
```

Lưu ý:

- Nếu là `python2.x` là mặc định thì thay `python` thành `python3`.
- Khi `server` khởi động sẽ có một tràng thông báo từ `log module`, `logging` ghi thẳng lên `stderr` nên sẽ có màu đỏ giống lỗi, vui lòng đừng sợ mà bỏ chạy.
- Khi tắt `server`, một số biến của thư viện `twisted` chưa được `release` tức thì, thay vì cố gắng restart `server` nhiều lần, hãy chờ một lúc rồi khởi động trở lại.

Khi có thông báo `Server has been bind at <host>:<port>` nghĩa là `server` đã sẵn sàng.

Sau đó chúng ta chạy `client` lên bằng lệnh:

```
python client.py
```

`Client` sẽ kết nối đến `server` và in ra lời chào mừng. Tới đây có thể gửi truy vấn `--help` để biết thêm thông tin chi tiết.

** Vẫn còn một tồn đọng là `server` của em không tắt bình thường bằng `^C` được, cách thủ công là `kill process` đó luôn. Em có tham khảo rằng dùng `docker` để serve nó lên rồi khi tắt thì tắt cái trong `docker` đi là xong.

Lời cảm ơn

Gửi lời cảm ơn đến những người thầy đã giao bài tập này cho em:

- Thầy **Lê Hà Minh**, giảng viên phụ trách bộ môn Mạng Máy Tính.
- Thầy **Nguyễn Thanh Quân**, giảng viên hướng dẫn thực hành và là người trực tiếp hướng dẫn thực hiện bài tập.

> Chúc mọi người có nhiều sức khỏe và thành công trong công việc.