Programming Assignment #5

Due : 21th Dec. (Fri),5:59 PM

1. Introduction

이번 과제는 멀티 쓰레드 데이터베이스 서버와 클라이언트를 제작하는 것이다. 이번 과제를 수행함으로써 소켓 프로그래밍과 쓰레드 프로그래밍을 연습하고, 서버와 클라이언트 간 동작 원리에 대해 배울 수 있다.

2. Problem specification

지금까지 구현하였던 DB의 기능을 서버와 클라이언트로 나누어 수행하게 된다.

클라이언트에서는 서버에 접속한 후 원하는 key의 value를 요청하거나 혹은 value의 수정을 요청할 수 있다.

서버에서는 클라이언트가 요구하는 행동을 수행하고 결과를 소켓을 통해 클라이언트에게 전송한다. 최대 연결 가능 client 수를 제한해야 하며, 이를 초과하는 client의 요청은 받지 않는다.

2.1. Client

- 클라이언트는 처음 실행할 시 ip address와 port 주소를 argument로 받는다.

```
./client 127.0.0.1 8888
```

- 현재 서버에 접속한 클라이언트 수가 서버가 수용할 수 있는 클라이언트 수를 초과하였을 경우 "Too many clients"를 출력한다.

2.2. Server

- 서버는 처음 실행 시 port와 최대 클라이언트 수, 그리고 hash table size를 입력 받는다.

```
./server 8888 6 100
```

- 서버가 이미 최대 클라이언트 수만큼 클라이언트들과 연결되어 있을 때는 추가로 새로운 클라이언트의 요청을 받을 수 없다.
- 서버는 클라이언트에서 받은 명령어들을 db.c에 구현한 db_get, db_put 등을 이용해 처리한다.
- 서버가 hash table과 file data를 다루는 방식은 이전 과제까지의 방식을 따른다.
- 단, value가 정수 이외 문자열이 들어올 수 있다는 점에 유의한다.
- input으로 주어질 key와 value의 크기는 각각 1KB를 넘지 않는다.

2.3. Protocol

서버와 클라이언트간 전송할 수 있는 메세지들의 종류는 다음과 같다. 이외의 명령이 전달될 때에는 UNDEFINED PROTOCOL을 출력한다.

1) CONNECT

클라이언트는 서버에 처음 접속할 시 서버에 "CONNECT" 메세지를 전송한다.

```
CONNECT ↵
CONNECT_OK ↵
```

이 메세지를 받으면 서버는 연결이 성립되었음을 알리는 뜻으로 CONNECT_OK를 클라이언트에게 전송한다.

2) DISCONNECT

클라이언트는 서버와의 연결을 종료할 시 서버에 "DISCONNECT" 메세지를 전송한다.

```
DISCONNECT ↵
BYE ↵
```

클라이언트는 서버로부터 "BYE" 라는 메세지를 받은 후 종료한다.

3) GET

GET은 서버로부터 특정 key의 value를 요구하는 프로토콜이다.

클라이언트에서는 다음과 같은 형태로 메시지를 전송한다.

```
GET [key] ↵
```

서버에서는 이를 받으면 먼저 [key]에 대한 [value]를 찾은 후 다음과 같은 형태로 클라이언트에게 응답한다.

```
GETOK [key] [value] ↵
```

만약 요청하는 key에 대한 value가 없을 경우 다음과 같이 응답한다.

```
GETINV ↵
```

4) PUT

PUT은 서버에게 특정 key에 대한 value를 추가 혹은 수정하도록 요구하는 프로토콜이다.

```
PUT [key] [value] ↵
```

서버는 이에 대해 다음과 같이 응답한다.

```
PUTOK ↵
```

2.3 (*) Asynchronous API

- 기존의 API는 모두 synchronous 하게 동작하였다. 즉, 어떤 요청을 보내면 그 요청에 대한 응답을 확실히 얻은 후에 다음 작동을 수행할 수 있었다. Asynchronous API는 이와 다르게, 요청을 보낸 후 응답 도착 여부와 관계없이 다음 작업을 수행할 수 있다.

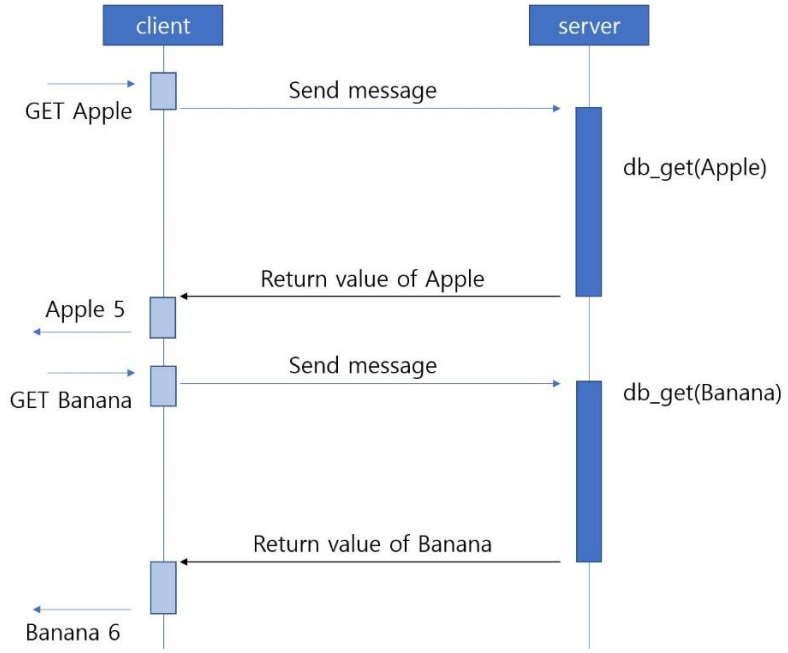

그림 1 Synchronous API

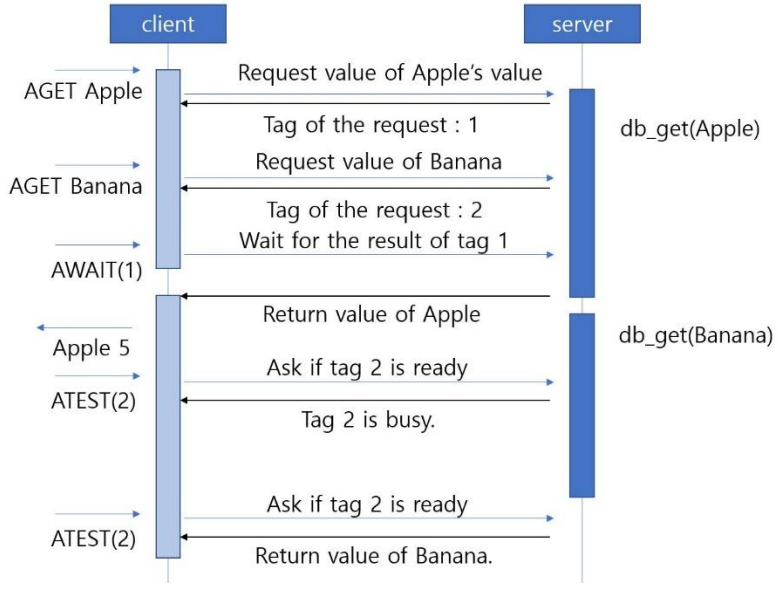

그림 2 Asynchronous API

추가해야 할 프로토콜은 다음과 같다.

1) AGET

서버에게 asynchronous 하게 GET을 요청한다. 서버에게는 다음과 같은 메시지를 전송한다.

AGET [key] ↵

이를 받으면 서버는 리퀘스트 마다 int형의 tag를 부여하고 다음과 같이 즉시 응답한다.

AGETOK [tag] ↵

그 후 서버는 요청받은 key에 대한 value를 찾는다.

2) AWAIT

Asynchronous하게 보낸 요청의 응답을 기다린다.

AWAIT [tag] ↵

클라이언트는 위 메시지를 전송한 후 서버로부터 아래와 같은 응답이 오기 전까지 대기한다.

서버는 이와 같은 명령을 받으면 해당 tag의 요청에서 요구한 value를 다음과 같이 전송한다.

GETOK [key] [value] ↵

만약 클라이언트로부터 온 메시지의 tag에 해당하는 요청이 없을 경우 잘못된 tag값임을 알리는 다음과 같은 메시지를 클라이언트에게 보내 응답한다.

AINV ↵

3) ATEST

클라이언트는 해당 tag의 요청의 완료 여부를 확인하는 메시지를 보낼 수 있다.

ATEST [tag]

만약 [tag]에 해당하는 요청이 아직 완료되지 않았다면 서버는 다음과 같이 응답한다.

[tag]_BUSY

[tag]에 해당하는 요청이 없을 경우 AWAIT에서와 같이 AINV를 전송한다.

[tag]에 해당하는 요청이 완료된 상태일 경우 역시 AWAIT에서와 같이 GETOK를 통해 결과를 전송한다.

Ex)

Client	Server
AGET Apple ATEST 1 ATEST 1 AWAIT 1 ATEST 1000	 AGETOK 1 1_BUSY 1_BUSY GETOK Apple 5 AINV

3. Restrictions

- 클라이언트와 서버 어플리케이션 코드는 각자 client.c와 server.c에 구현한다.
- Makefile 수정을 통한 추가적인 코드 구현을 허용한다.
- 과제 구현을 위해 여태까지 배운 리눅스 시스템 콜/라이브러리 함수를 이용한다.
- 어떤 자원을 동적으로 할당 받았다면, 프로그램 종료 전에 반드시 해제해야 한다.
  - 여기서 자원이란 파일, 메모리, 자식 프로세스를 뜻한다.

4. Hand in instruction

- 작성한 코드 상단의 주석에 이름과 학번을 작성한다.
- **make 명령을 통해 프로그램이 제대로 만들어지는지 확인한다.**
  - **제출한 코드가 컴파일되지 않을 경우, 경우에 따라서 해당 제출은 채점되지 않거나 심한 페널티를 받게 될 것이다.**
- 프로그램 코드를 "학번.tar.gz" 형태로 압축한다.
- 본 과제 수행 시 구현 방법과 디자인을 설명하는 보고서를 PDF 포맷으로 작성하여 "학번.pdf" 이란 이름을 붙인다

5. Logistics

- 본 과제는 혼자 수행한다.
- 과제 제출 시간은 메일 도착 시간을 기준으로 하며, 과제를 지연 제출하면 **기한 직후**엔 10%가 감점되고, **매 24시간마다** 10%씩 추가로 감점된다.
- **Copy 할 경우, 연구실 자체 규정에 따라 처벌하며, 상당한 불이익이 있을 수 있다.**

Have fun!
