System Software Experiment 2

README REPORT

Asynchronous DataBase Socket Programming

2014313299 박세진

My database 구조

내 데이터베이스의 구조는 기존의 LAB과 다른 파일 구조를 설계했다. 이번 서버의 데이터베이스는 최초 데이터베이스 실행 시 1024개의 빈 key파일이 생성되고 클라이언트로부터

명령어를 수행하면서 이 key 파일에 Key 데이터를 저장하고 이 key 데이터에 해당하는 value값은 새로 생성해 val 파일로 관리한다.

Server - Client 통신 구조

Server와 Client가 통신하는 형식은 다음과 같다.

1. Server가 Client를 위한 소켓 연결을 대기하고 있는다.
2. Client가 Socket을 생성하고 Server의 IP 주소와 Port 번호로 연결을 시도한다.
3. Client가 CONNECT 입력을 받으면 Server에게 전송해 Server가 현재 통신을 연결 성공 또는 실패를 Client에게 응답해준다.
4. 연결이 설정된 이후 Client가 사용자로부터 받은 입력을 Server에 전송하고 Server는 이 전송에 대한 작동을 한 뒤, 작동에 대한 응답을 Client에게 전송하고 Client는 이 전송을 받아 화면에 응답 메시지를 출력해준다.
5. 4번의 과정을 파일이 종료되거나 사용자가 DISCONNECT 명령어를 입력했을 시에만 Server에 종료를 요청하고 Client의 통신 스레드를 종료하게 한다. 마찬가지로 Server에서도 이 Client에게 할당되어있었던 스레드를 종료시키고 또 다른 Client의 연결을 대기한다.

Server - database 접근

Server는 Client에게 메모리 상의 데이터 또는 파일 내부의 데이터에서 요청한 Key 값에 맞는 Value를 찾아 key value 쌍을 반환해주는 GET 요청과 특정 key와 mapping 되는 value값을 설정하는 PUT 요청으로 현재 생성된 database에 접근할 수 있다. (Asynchronous 통신에서는 AGET 요청이 database에 접근한다.)

Server에서 입력받은 size만큼의 database를 생성하고 get 요청 시 db_get 함수를 실행시키는데 여기서 같은 hash값을 가지는 key를 찾는 것에 대해서는 mtx라는 pthread_mutex_t 타입의 배열으로 lock을 걸어서 메모리 탐색을 하고, 메모리 탐색이 끝난 후에는 unlock한 후에 파일 탐색을 시작한다. 여기서 파일 탐색에 lock을 하지 않은 이유는 여러 client들이 한 파일을

read하는 것에 대해서는 문제가 발생하지 않고 write할 때 read하는 것이 문제를 발생시키는데 메모리의 database를 파일로 내릴 때 이 문제를 처리했기 때문에 lock을 걸지 않고 처리한다.

Client가 put 요청을 하면 일단 메모리에 key - value쌍을 올리고 현재 database 의 수를 증가시킨다. 만약 database의 key-value 쌍의 수가 database size와 같아질 때, cnct_mutex와 cnct_cond라는 pthread_mutex_t 타입 변수, pthread_cond_t 타입 변수를 이용해 모든 client가 이 상황에 도달했을 때 가장 마지막에 이 상황에 도달한 Client가 메모리에 있는 모든 database key-value 쌍을 파일로 내리는 역할을 맡도록 했다. 파일을 모두 내린 후에는 wait하고 있던 다른 client들을 풀어줌으로써 파일 접근에 있어서 충돌을 피하게 했다.

Server - AGET 처리

AGET 처리를 위해 구조체 async_t를 선언해서 이 구조체 배열 Asynn을 이용해 모든 client는 최대 256(MAX_ASYNC)개의 비동기 요청을 사용할 수 있도록 했다. 1부터 256까지의 tag를 앞부터 순서대로 AGET 요청에 대해 부여해주는데 여기에는 현재 어떤 클라이언트가 요청한 것인지를 기억하기 위해 cfd라는 client와 통신하는 socket의 file descriptor를 적어놓는다. ATEST나 AWAIT으로 이 값을 찾을 때 같은 socket의 file descriptor를 가지지않은 client의 요청은 AINV로 응답하기 위함이다.

사용한 lock 방식 - pthread_mutex_lock

pthread_mutex_t 변수

1. mtx : 이 mutex 변수는 각 database의 hash 값에 대응되는 mutex로 같은 hash 값을 가지는 key를 database에서 찾거나 value를 저장할 때 database 내에서 read 또는 write 간의 충돌을 막기 위한 lock으로 사용한다.
2. cnct_mutex : db를 파일로 내릴 때 이 파일을 내리는 것을 한 Client가 전담해서 내리고 나머지 Client는 이 Client가 파일을 다 내릴 때까지 cnct_cond 변수와 함께 사용하여

wait와 lock을 한다. 또한 현재 서버에 접속해있는 Client의 수를 저장하는 전역변수인 client_n의 값을 변경시킬 때에도 사용한다.

3. kv_mutex : 전역 변수 kv_s (현재 메모리의 database에 들어있는 key-value 쌍의 개수) 에 접근할 때 충돌이 일어나지 않도록 하기 위한 lock을 걸기 위한 변수이다.
4. asynn_mutex : client들의 AGET 요청 시 MAX_ASYNC(현재 256개)의 크기를 가지고 있는 Asynn 구조체 배열에 접근하기 위한 mutex lock 변수이다.

pthread_cond_t 변수

1. cnct_cond : cnct_mutex와 함께 db를 파일로 내릴 때 한 Client만 제외하고 나머지 Client들을 wait시켜주기 위한 cond 변수이다.
2. asynn_cond : 이 변수는 Client의 AWAIT 요청 시 만약 서버가 해당 tag의 db_get을 처리하고 있는 중이라면 현재 Asynn 구조체 배열을 잡고 있는 lock을 풀어줘서 다른 Client가 사용할 수 있도록 wait함으로써 db_get을 모두 처리 할 때까지 기다리도록 하는 cond 변수이다.

Finally

이번 SSE2의 과제를 통해 내가 구현한 DB는 겉으로 드러나는 파일 저장 또는 파일 탐색의 형태가 Server가 Client에게 어떤 요청이든 즉시 응답을 한 후 다시 Client가 요청하는 방식으로 이루어지기 때문에 Synchronous한 형태이지만 AGET을 통해 새로운 thread를 생성하여 db_get 함수를 실행하게 하여 한번 더 이 key에 대한 value 값을 요청했을 때 더 빠르게 256(MAX_ASYNC)개의 다른 메모리를 통해 빠르게 찾을 수 있다.