목차

[◎ 목적 및 조건 2](#_Toc151647295)

[◎ 개발 순서 및 과정 8](#_Toc151647296)

[1. Server, Client, TCP 통신 개발 8](#_Toc151647297)

[2. 스레드 풀 적용 (Queue, 템플릿) 9](#_Toc151647298)

[3. 메시지 큐 생성 (Queue, 템플릿) 10](#_Toc151647299)

[4. 수집 데이터 형식 변경 12](#_Toc151647300)

[5. 데이터 MariaDB 저장 12](#_Toc151647301)

[6. 최적화 13](#_Toc151647302)

[◎ 설계서 및 설계도면 17](#_Toc151647303)

[◎ 비동기 구현 18](#_Toc151647304)

[◎ 예상 결과 19](#_Toc151647305)

[◎ 각종 샘플 20](#_Toc151647306)

[Server 설계 20](#_Toc151647307)

[Client 설계 22](#_Toc151647308)

[MariaDB 사용 24](#_Toc151647309)

[CPU 설계 26](#_Toc151647310)

[NIC 설계 28](#_Toc151647311)

[메모리 설계 31](#_Toc151647312)

[Disk 설계 33](#_Toc151647313)

[Json 설계 36](#_Toc151647314)

# ◎ 목적 및 조건

목적 : 정보를 수집하는 PC의 Service와 데이터를 수집 저장하는 Server의 Service를 각각 개발한다.

전체 기능 컴파일 : g++ -o test test.cpp -lboost\_system -I/usr/include/ -L/usr/lib64 -lmysqlclient -ljsoncpp

조건

1. Rocky Linux(록키 리눅스)를 사용한다.

* CentOS가 up Stream으로 전환되어 보안에 취약해졌기 때문이다.

1. C++ 언어를 사용한다.

* **$sudo yum install -y gcc-c++** // C++ 설치
* **$g++ –version** // 설치 및 버전 확인
* **$vim 파일명.gcc** // C++ 파일 작성

1. Maria DB를 사용한다.

* Maria DB는 MySQL에서 파생되어 탄생한 오픈 소스 관계형 데이터베이스 시스템(RDBMS)이다. MySQL과 공통점이 많고 호환성도 높다.
* 컬럼형 DB 엔진을 사용하고 있어 열 단위로 데이터를 처리한다. 동일한 형의 데이터가 일렬로 세우기 위하여 데이터의 압축 효율이 높아지며, 일반적으로 열 방향에서 집계 처리도 빨라지고 열마다 더 분산되어 병렬로 수행할 수 있다는 장점을 가지고 있다.
* **$sudo dnf install –y mariadb-server** // 마리아 DB 설치
* **$sudo systemctl start mariadb** // 마리아 DB 실행
* **$sudo systemctl status mariadb** // 서비스 상태 확인
* **$sudo systemctl enable mariadb** // 부팅 시 Maria DB 시작 확인
* **$sudo mysql\_secure\_installation** // 보안 스크립트를 실행
* **$mysqladmin -u root -p version** // 설치 테스트
* **$mysql -u root –p** // DB에 접속(root 계정)
* **$ -I/usr/inclued/ -L/usr/lib64 –lmysqlclient** // mysql 컴파일
* Maria DB 설치 참고 블로그 (<https://ko.linux-console.net/?p=6555>)
* MariaDB와 MySQL의 기능적 차이점(<https://aws.amazon.com/ko/compare/the-difference-between-mariadb-vs-mysql/>)
  + JSON 데이터 처리  
    MariaDB와 MySQL은 JSON 데이터의 검색 및 저장을 지원한다. 하지만 JSON 보고서를 서로 다른 방식으로 저장한다. MariaDB는 JSON 보고서를 문자열로 저장하는 반면, MySQL은 JSON 보고서를 바이너리 객체로 저장한다.  
    또한 MySQL과 MariaDB 모두 지원하지 않는 JSON 함수가 있다. MariaDB는 JSON\_QUERY와 JSON\_EXISTS를 지원하지만 MySQL은 지원하지 않는다. 마찬가지로, MySQL은 MariaDB에서는 지원하지 않는 네이티브 JSON 데이터 유형 JSON\_TABLE을 지원한다.
  + 기능  
    MySQL은 단일 열에 여러 데이터 값을 정의하고 함수를 사용하여 열을 수정할 수 있는 동적 열을 제공한다. 동적 열을 사용하면 데이터 마스킹을 통해 민감한 정보를 보호할 수 있다.  
    반면, MariaDB는 데이터베이스 보기에서 보이지 않는 열을 지원한다. 사용자가 SELECT 문을 수행하거나 INSERT 문에서 값을 요청할 때는 보이지 않는 열이 나열되지 않는다.
  + 암호화  
    MySQL을 사용하면 관리자가 다시 실행 및 실행 취소 로그를 구성하고 암호화할 수 있지만, MySQL은 임시 테이블스페이스나 바이너리 로그는 암호화하지 않는다.  
    반면 MariaDB는 바이너리 로그 암호화와 임시 테이블 암호화를 지원한다.
  + 스레드 풀링  
    MariaDB의 경우 Community Edition의 일부인 스레드 풀 플러그인에 스레드 풀링이 포함되어 있다. MariaDB는 한 번에 20만 개 이상의 연결을 관리할 수 있는 기능을 제공한다.  
    MySQL의 경우 Enterprise 버전에 스레드 풀 플러그인이 있다. 하지만 MariaDB만큼 많은 연결을 한 번 처리할 수는 없다.
* MariaDB와 MySQL의 문법 차이 (<https://norimoon.com/entry/MariaDB-vs-MySQL-%EB%AC%B8%EB%B2%95-%EC%B0%A8%EC%9D%B4>)
  + LIMIT절  
    LIMIT절은 쿼리에서 반환되는 행 수를 제한하는 데 사용된다. MariaDB에서OFFSET 키워드를 사용하여 시작 행을 지정할 수 있다.  
    SELECT \* FROM table LIMIT 10 OFFSET 5;  
    MySQL에서는 쉼표 구분을 사용하여 각 시작 행을 지정할 수 있다.  
    SELECT \* FROM table LIMIT 5, 10
  + Full-Text 검색  
    열 내에서 텍스트를 검색할 수 있는 검색 기능이다. MariaDB에서는 MATCH()함수를 사용하여 Full-Text 검색을 수행할 수 있다.  
    SELECT \* FROM table WHERE MATCH(column) AGAINST(‘search term’);  
    MySQL에서는 MATCH()함수를 AGAINST() 키워드와 함께 사용할 수 있습니다.  
    SELECT \* FROM table WHERE MATCH(column) AGAINST(‘search term’ IN NATUREAL LANGUAGE MODE);
  + 정규 표현식  
    MariaDB에서 REGEXP 연산자를 사용하여 정규식을 일치시킬 수 있다.  
    SELECT \* FROM table WHERE column REGEXP ‘pattern’’  
    MySQL에서는 REGEXP 연산자 또는 RLIKE 연산자를 사용할 수 있다.  
    SELECT \* FROM table WHERE column REGEXP ‘pattern’;  
    SELECT \* FROM table WHERE column RLIKE ‘pattern’;
  + 날짜 및 시간 함수
  + Window 함수  
    MariaDB에서는 OVER 절을 사용하여 window를 지정할 수 있다.  
    SELECT column, SUM(column) OVER (PARTITION BY other\_column) FROM table;  
    MySQL에서는 WINDOW 절을 사용하여 window를 지정할 수 있다.  
    SELECT column, SUM(column) OVER (PARTITION BY other\_column) WINDOW w1 AS (ORDER BY column) FROM table;
  + Join 절  
    MariaDB에서는 USING절을 사용하여 같은 이름의 열을 조인할 수 있다.  
    SELECT \* FROM table1 JOIN table2 USING (column);  
    MySQL에서는 USING 절을 사용할 수도 있지만 ON 절을 사용하여 이름이 다른 열에 조인할 수도 있다.  
    SELECT \* FROM table1 JOIN table2 ON table1.colum = table2.other\_column;

1. 비동기(Asynchronous) 형식을 사용한다.
   * 병렬적으로 작업을 수행하는 방식이다.
   * 비동기는 요청한 작업에 대하여 완료 여부를 신경쓰지 않고 자신의 그 다음 작업을 수행한다. I/O 작업과 같은 느린 작업이 발생할 때, 기다리지 않고 다른 작업을 처리하면서 동시에 처리하여 멀티 작업을 진행할 수 있기 때문에 전반적인 시스템 성능 향상에 도움을 준다.
   * 작업을 여러 개 요청했을 때 순서를 보장할 수 없다.
   * 요청과 응답이 분리되어 있기 때문에, 에러가 발생했을 때 어떤 요청에서 에러가 발생했는지 파악하고 처리하는 것이 쉽지 않다.
   * 리눅스에서 비동기 라이브러리는 여러가지 있지만 그 중 Boost.Asio, cppcoro, libuv, folly, asio가 존재한다. 아래는 각 라이브러리의 장단점이다.
     + **Boost.Asio 장점**  
       1) 다양한 플랫폼에서 사용 가능하며, Boost 라이브러리의 일부로서 안정성이 높다.  
       2) 다양한 프로토콜 및 I/O 작업을 지원하며, 풍부한 기능을 제공한다.  
       **Boost.Asio 단점**  
       1) 학습 곡선이 상대적으로 가파르며, 코드가 다소 복잡할 수 있다.
     + **cppcore 장점**  
       1) C++20의 코루틴을 사용하여 비동기 코드를 간결하게 작성할 수 있다.  
       2) 간단하면서도 효율적인 비동기 코드를 작성할 수 있는 장점이 있다  
       **cppcore 단점**  
       1) 아직 C++20이 완전히 채택되지 않은 프로젝트에서는 사용이 제한될 수 있다
     + **libuv 장점**  
       1) Cross-platform 지원으로 여러 플랫폼에서 동작한다.  
       2) Node.js와 같이 대규모 서버 애플리케이션에서 실전 경험이 있는 라이브러리이다.  
       **libuv 단점**  
       1) C 언어로 작성되어 있어서 C++ 특유의 기능을 활용하기 어려울 수 있다.
     + **folly 장점**  
       1) Facebook에서 사용 중인 높은 성능을 제공하며, 많은 고급 기능을 지원한다.  
       2) futures, promises, coroutines와 같은 강력한 도구를 제공한다.  
       **folly 단점**  
       1) 외부 의존성이 있는 경우가 있으며, 학습 곡선이 다소 가팔라질 수 있다.
     + **asio 장점**  
       1) Boost.Asio의 스탠드얼론 버전으로, Boost 의존성이 없어서 더 가볍다.  
       2) 비교적 단순하고 직관적인 API를 제공한다.  
       **asio 단점**  
       1) Boost.Asio에 비해 일부 기능이 제한적일 수 있으며, 개발이 활발하지 않을 수 있다.
     + 이 중 안정성이 높은 Boost.Asio를 사용하려고 한다. 만약 더 빠른 라이브러리가 필요하다면 Asio2, asio-standalone, seastar를 고려할 수 있지만 속도를 제외한 다른 성능은 Boost.Asio에 비해 밀릴 수 있다.  
       (Boost.Asio 참고 <https://www.boost.org/doc/libs/1_83_0/doc/html/boost_asio/index.html>)

* Boost.Asio 설치 방법  
  **$ sudo dnf install boost-devel** // Boost.Asio 설치  
  **$ g++ -o [파일명] [파일명.cpp]-lboost\_system** // boost.asio 컴파일

1. 전송 데이터 XML(혹은 JSON) 형식으로 개발한다.

* XML파서 라이브러리 종류 : 여러가지 라이브러리 중 PugiXML, TinyXML2, RapidXML, Boost.PropertyTree, Xcrces-C++ 등이 존재한다. 각자 아래와 같은 장단점을 가지고 있다.
  + **PugiXML의 장점**1) 뛰어난 성능과 빠른 파싱 속도를 제공한다.  
    2) 간결하고 직관적인 API를 통해 사용이 용이하다.  
    3) XPath 지원으로 XML 문서 내에서 특정 노드를 선택하는 데 용이하다. **PugiXML의 단점**1) C++11 이상을 필요로 하는데, C++11 이전의 환경에서는 사용에 제약이 있을 수 있다.
  + **TinyXML2의 장점**1) 가볍고 단일 헤더 파일로 간편한 통합이 가능하다.  
    2) 유연하면서도 간단한 인터페이스를 제공한다. **TinyXML2의 단점**1) 성능 면에서는 PugiXML에 비해 상대적으로 떨어질 수 있다.  
    2) 멀티바이트 문자열 처리에 제한이 있을 수 있다.
  + **RapidXML의 장점**1) 빠른 파싱 속도와 낮은 메모리 사용량을 자랑한다.  
    2) 단일 헤더 파일로 구성되어 있어 간편하게 사용할 수 있다. **RapidXML의 단점**1) 문서 수정이 다소 제한적일 수 있다.  
    2) XPath 지원이 부족하며, 기능적으로 간소한 부분이 있을 수 있다.
  + **Boost.PropertyTree의 장점**1) Boost 라이브러리의 일부로서 안정성과 신뢰성이 높다.  
    2) 다양한 데이터 형식과 함께 XML과 JSON을 처리할 수 있는 범용성이 있다. **Boost.PropertyTree의 단점**1) 성능 면에서는 다른 라이브러리들에 비해 상대적으로 느릴 수 있다.  
    2) Boost 라이브러리를 사용하는 데 익숙하지 않은 경우 학습 곡선이 있을 수 있다.
  + **Xcrces-C++의 장점**1) Apache Software Foundation에서 제공하는 안정적이고 강력한 라이브러리다.  
    2) 다양한 기능과 옵션을 제공하여 복잡한 XML 작업을 다룰 수 있다. **Xcrces-C++의 단점**1) 무거운 라이브러리로 인해 경량 애플리케이션에는 적합하지 않을 수 있다.  
    2) 초기 설정 및 사용이 다른 라이브러리에 비해 복잡할 수 있다.
* Json 라이브러리 종류 : 많은 라이브러리 중 주로 사용되는 것은 nlohmann/json, jsoncpp, RapidJSON가 있다. 세 가지 라이브러리는 아래와 같은 장단점을 가지고 있다.
  + **nlohmann/json의 장점**  
    1) 단일 헤더 파일로 구성되어 있어 간단하게 사용할 수 있다.  
    2) Modern C++의 기능을 활용하며 사용자 친화적인 API를 제공한다.  
    3) jsoncpp에 비해 전반적인 성능, 코드사이즈, 속도 등이 우수하다.  
    **nlohmann/json의 단점**  
    1) 헤더 파일의 크기가 상대적으로 크기 때문에 프로젝트의 크기가 큰 경우 빌드 시간이 길어질 수 있다.
  + **jsoncpp의 장점**  
    1) 간단하고 사용하기 쉬운 API를 제공한다.  
    2) 소스 코드가 C++로 작성되어 있어 쉽게 이해할 수 있고 수정할 수 있다.  
    **jsoncpp의 단점**  
    1) 라이브러리의 유연성이 부족할 수 있다.  
    2) 최신 C++ 표준과 어느 정도 떨어져 있을 수 있다.
  + **RapidJSON의 장점**  
    1) 빠른 속도와 낮은 메모리 사용량을 자랑한다  
    2) SAX 스타일의 파싱을 지원하여 대용량 JSON 데이터에 효과적이다.  
    **RapidJSON의 단점**  
    1) 사용하기에 다소 복잡할 수 있고, 초기 학습 곡선이 존재한다.  
    2) 다른 몇 가지 라이브러리에 비해 사용자 친화성이 낮을 수 있다.
  + 이 세가지 중 JSON 라이브러리를 처음 사용해보기 때문에 사용하기 쉬우면서도 jsoncpp보다는 성능이 좋은 nlohmann/json 라이브러리를 사용한다.
  + nlohmann/json.hpp 상세 설명 (<https://csj000714.tistory.com/430>)
  + Rocky Linux에서 nlohmann/json 설치 방법  
    **$ sudo yum install cmake  
    $ git clone https://github.com/nlohmann/json.git  
    $ cd json  
    $ mkdir build  
    $ cd build  
    $ cmake ..  
    $ make  
    $ sudo make install**
  + **$ g++ [파일명.cpp] -o [파일명] -std=c++11 –ljsoncpp** // json 실행파일 생성

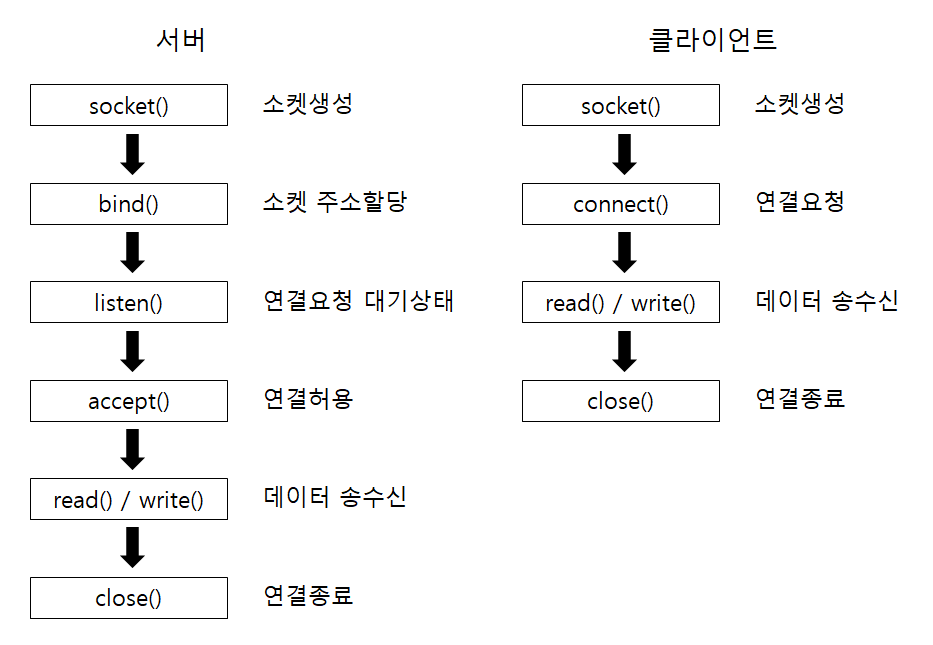
1. Client에서 주기적으로 단말의 정보를 수집하여 TCP로 Server에 접속해 정보를 전송한다.

* 클라이언트가 데이터를 송신 후 서버와 클라이언트 간 연결이 종료된다.

1. CPU, 메모리, Disk, NIC의 정보를 수집하여 전송한다.
   * CPU의 수집 정보
     + CPU 코어 수
     + CPU 모델 이름
     + CPU 아키텍처
   * NIC 수집 정보
     + 운영 상태 정보
     + 속도 정보
     + MAC 주소
   * 메모리 수집 정보
     + 메모리에 관한 모든 정보
   * Disk 수집 정보
     + 디스크 장치 이름
     + 완료한 입출력 작업의 수
     + 병합된 입출력 작업의 수
     + 총 섹터 수
     + 읽기 작업에 소요된 시간(ms)
     + 쓰기 작업에 소요된 시간(ms)

# ◎ 개발 순서 및 과정

## 1. Server, Client, TCP 통신 개발

* Server Service
  + Boost.Asio 라이브러리를 사용하여 비동기 형식을 구축한다.  
    **$ sudo dnf install -y boost-devel** // boost 라이브러리 설치  
    **#include <boost/asio.hpp>** // asio를 사용하기 위한 헤더  
    **$ g++ [programname.cpp] -o [programname] -lboost\_system** // 컴파일 방법
  + 여러 클라이언트의 연결을 동시에 처리하기 위하여 스레드 풀 사용한다. 성능을 향상시키는 결과를 가져올 수 있다.
  + 메시지 큐를 사용하여 스레드 풀에서 메시지를 처리한다. 이를 통해 비동기적으로 메시지를 처리할 수 있다.
  + 클라이언트에게 받은 XML(혹은 JSON)형식의 정보를 MariaDB에 저장한다.
* Client Service
  + Boost.Asio 라이브러리를 사용하여 비동기 형식을 구축한다.
  + CPU(Core 별), NIC, 메모리, Disk 정보 수집하여 XML(또는 JSON)형식으로 파일을 생성한다.
  + 생성한 파일을 서버에게 송신 후 파일을 삭제하고 소켓을 닫아 서버와의 연결을 종료한다..
* TCP 통신  
  TCP 의 구현은 아래 사진과 같이 이뤄진다.  
    
  통신이 이뤄지는 것의 순서를 정리해보면 아래와 같다.

|  |  |
| --- | --- |
| 서버 | 클라이언트 |
| 1. 기다리는 소켓(listening socket) 생성 후 클라이언트 연결 대기 |  |
|  | 2. 클라이언트 소켓 생성 후 서버 연결 시도 |
| 3. 클라이언트 연결 시도 승인 |  |
| 4. 자료 수신 | 4. 자료 송신 |
| 5. 연결 종료 | 5. 연결 종료 |

다만, Boost.Asio를 사용하게 된다면 TCP 통신은 아래와 같이 이뤄진다.

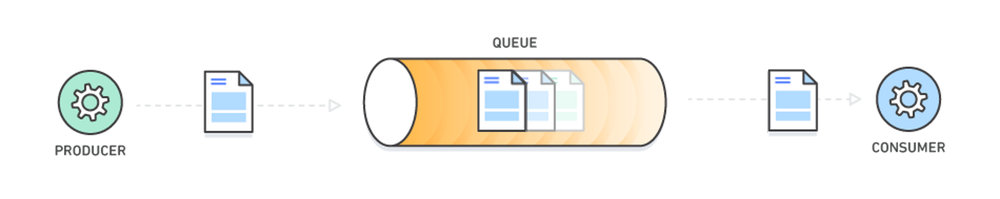
* + 소켓 생성 및 연결 : ip::tcp::acceptor 및 ip::tcp::socket 클래스를 사용하여 생성 및 연결한다.
  + 연결 요청 대기 및 수락 : async\_accept()를 사용하여 비동기적으로 연결을 수락한다.
  + 데이터 수신 및 송신: async\_read\_until() 함수를 사용하여 비동기적으로 데이터를 수신하고, 메시지큐를 통해 스레드 풀에서 메시지를 송신한다.
  + 소켓 닫기: 소켓은 라이프타임 관리를 통해 자동으로 닫힌다.

## 2. 스레드 풀 적용 (Queue, 템플릿)

* 스레드 풀 : 특정 함수를 실행하는 Worker 스레드를 관리하는 객체 (<https://narakit.tistory.com/141>)
  + 요청받은 작업을 작업 큐에 삽입
  + 대기 중인 워커 스레드중 하나를 wakeup하여 작업을 큐에서 꺼내고 실행
  + 다수의 스레드에서 접근하는 작업 큐를 뮤텍스로 보호
* 스레드 풀의 이점
  + 동시성 및 병렬성 : 스레드 풀을 사용하면 여러 클라이언트와의 연결 간에 동시에 작업을 수행할 수 있다. 이는 여러 클라이언트가 동시에 메시지를 보내고 수신할 때, 각각의 작업을 병렬로 처리하여 응답 시간을 개선할 수 있다.
  + 응답성 향상 : 스레드 풀을 사용하면 주 스레드(main thread)가 블록되지 않고 다른 작업을 수행할 수 있으므로 서버 응답성이 향상된다. 새로운 클라이언트 연결을 수락하거나 기존 클라이언트로부터 메시지를 읽어오는 작업이 별도의 스레드에서 비동기적으로 처리된다.
  + 자원 효율성 : 스레드 풀은 스레드를 효과적으로 재사용하므로 쓰레드 생성 및 소멸에 따르는 오버헤드를 감소시킨다. 이는 자원을 더 효율적으로 사용하게 도와준다.
  + 코드 간결성 : 비동기 작업을 수행하고 여러 클라이언트 간에 작업을 조정하는 코드를 간결하게 유지할 수 있다. 스레드 풀을 사용하면 복잡한 작업 스케줄링 및 동기화 문제에 대한 코드가 감소하고 가독성이 향상된다.
  + 스케일링 용이성 : 스레드 풀은 작업 부하가 증가할 때 쉽게 스케일링할 수 있는 구조를 제공한다. 즉, 더 많은 스레드를 추가함으로써 높은 부하에 대응할 수 있다.
* 스레드 풀의 단점
  + 복잡성 증가 : 스레드를 사용하면 프로그램의 복잡성이 증가할 수 있다. 스레드 간의 동기화와 데이터 공유에 관한 복잡한 문제를 다루어야 하므로, 코드 작성 및 유지보수가 어려워질 수 있다.
  + 데드락 및 경합 상태 가능성 : 스레드 간의 안전한 동시성을 보장하기 위해서는 데드락이나 경합 상태와 같은 다양한 복잡한 문제를 다루어야 한다. 잘못된 스레드 동기화는 예측 불가능한 버그를 초래할 수 있다.
  + 자원 소비 : 각 스레드는 운영체제에서 관리하는 자원을 소비한다. 따라서 너무 많은 스레드를 생성하면 시스템의 자원을 과도하게 소비하게 될 수 있다.
* 위와 같은 단점들을 해결하기 위해 메시지 큐를 병합하여 사용한다. 메시지 큐와 병합하며 얻는 이점은 아래와 같다.
  + 동기화 용이성 : 메시지 큐를 사용하면 명시적인 동기화가 더 적게 필요할 수 있다. 메시지 큐를 통해 데이터를 교환하는 데 필요한 동기화는 대개 메시지 큐 자체에서 처리되기 때문이다..
  + 간접 통신 : 메시지 큐를 사용하면 스레드 간 통신을 간접적으로 처리할 수 있다. 이는 직접적인 스레드 간 통신을 피하고, 데드락이나 경합 상태 같은 문제를 덜어줄 수 있다.
  + 메시지 큐의 관리 특성 : 메시지 큐는 스레드 풀에서 메시지를 처리하는 데 사용될 수 있으며, 작업 큐를 통해 비동기적으로 처리될 수 있다. 이는 작업 스케줄링 및 자원 관리에 유용할 수 있다.
  + 스레드 간 결합도 감소 : 메시지 큐를 사용하면 스레드 간 결합도가 감소하고, 각 스레드가 덜 의존적이 되어 각각 독립적으로 작동할 수 있다.
* 스레드 풀 사용 이유
  + 다중 클라이언트를 효과적으로 처리하고 서버의 응답성과 자원 효율성을 향상시키기 위해 사용하였다.

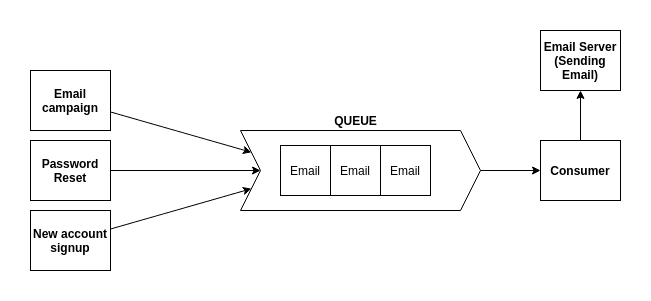
## 3. 메시지 큐 생성 (Queue, 템플릿)

* 프로세스 또는 프로그램 간에 데이터를 교환할 때 사용하는 통신 방법  
  (참고 블로그 : <https://tecoble.techcourse.co.kr/post/2021-09-19-message-queue/>)
* 메시지 지향 미들웨어(Message Oriented Middleware : MOM)를 구현한 시스템



* 소비자(Consumer)가 실제로 메시지를 어느 시점에 가져가서 처리하는 지는 보장하지 않음
  + 이메일 전송

1. 비밀번호 재설정을 위해 이메일을 발급하는 서비스, 회원가입을 위해 이메일을 발급하는 서비스 등은 메시지(이메일)를 큐에 넣을 수 있다.
2. 어느 서비스든 메시지 큐의 메시지를 하나씩 소비하고, 그저 이메일이 전송되어야 할 곳으로 이메일을 전송
3. 메시지 큐에 들어오는 메시지 수가 너무 많아지는 경우 이메일 전송 전용 서비스 인스턴스를 더 둠으로써 확장할 수 있음



* + 블로그 포스팅

1. 사용자가 고용량의 이미지가 포함된 블로그 포스팅을 한다.
2. 이미지는 저장소에 전송된다.
3. 업로드 된 이미지에 대한 정보가 포함된 메시지를 이미지 최적화 서비스의 메시지 큐에 담는다.
4. 이미지 최적화 서비스는 저장소에서 이미지를 가져와 최적화하고, 2번에서 저장해놨던 이미지를 대체한다.

* 메시지 큐의 이점
  + 비동기 : 생산된 메시지의 저장, 전송에 대해 동기화 처리를 진행하지 않고, 큐에 넣어 두기 때문에 나중에 처리 가능하다.
  + 낮은 결합도 : 생산자 서비스 혹은 소비자 서비스가 독립적으로 행동하게 됨으로써 서비스 간 결합도가 낮아진다.
  + 확장성 : 생산자 서비스 혹은 소비자 서비스를 원하는 대로 확장할 수 있다.
  + 탄력성 : 소비자 서비스가 다운되더라도 메시지는 메시지 큐에 남아있다. 소비자 서비스가 다시 시작될 때마다 추가 설정이나 작업을 수행하지 않고도 메시지 처리를 시작할 수 있다.
  + 보장성 : 큐에 보관되는 메시지가 소비자 서비스에 전달된다는 일반적인 보장을 제공한다.
* 메시지 큐의 사용 이유
  + 여러 스레드가 동시에 작동하므로 스레드 간의 통신이 필요한데, 메시지 큐를 사용하면 스레드 간에 데이터를 안전하게 전달할 수 있다.
  + 공유 데이터에 대한 접근을 조율하고 동기화를 처리하며, 이를 통해 스레드 간 경쟁조건 및 데드락을 방지할 수 있다.
  + 멀티스레드 환경에서 안전하게 동작할 수 있도록 도와준다.

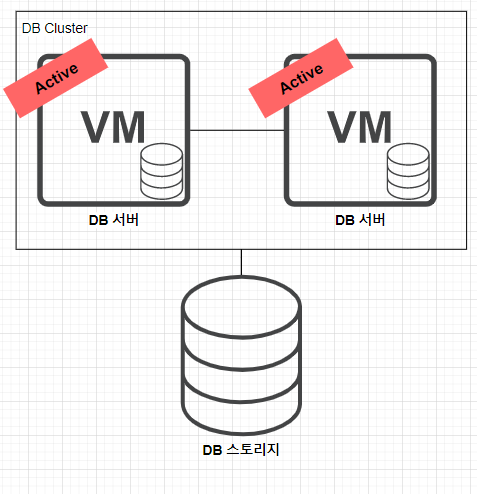
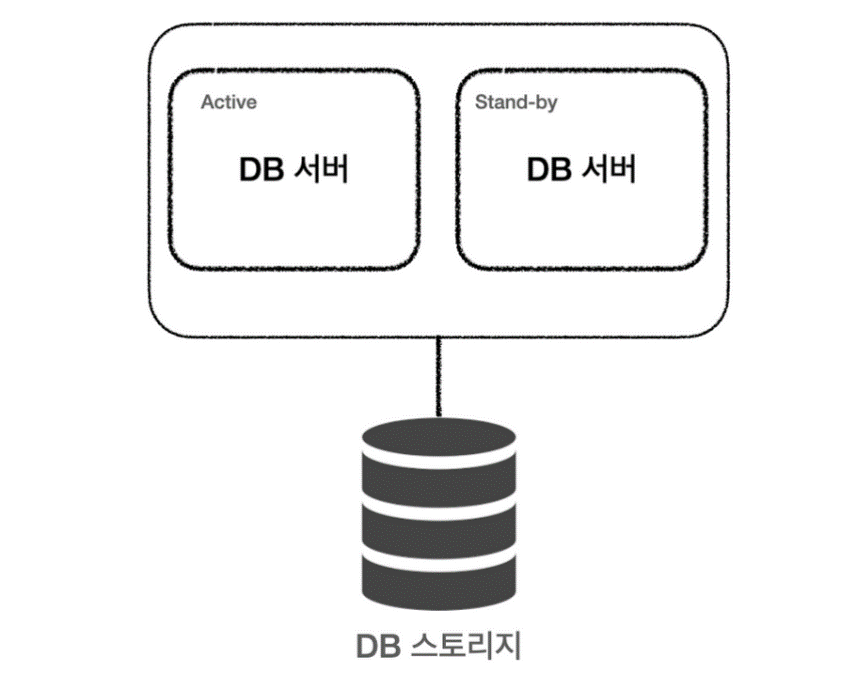
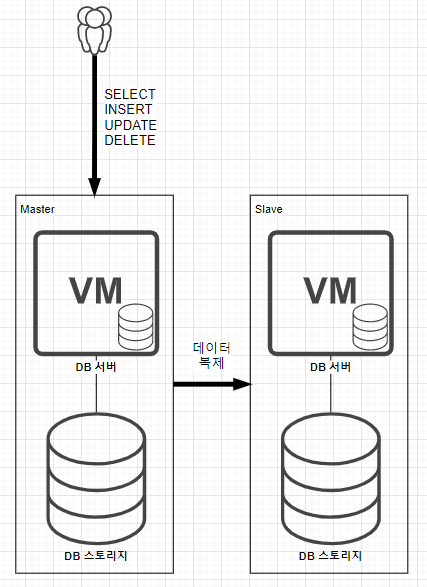
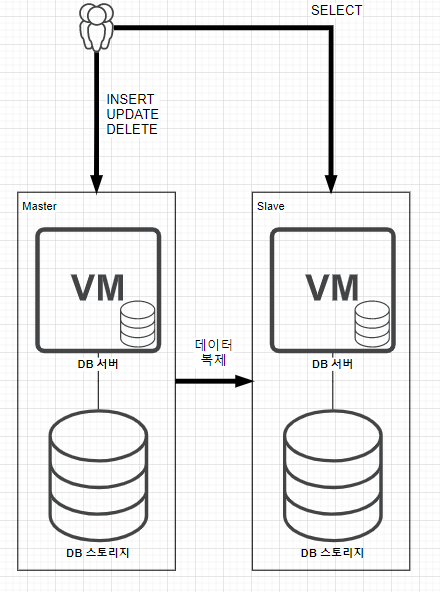
## 4. 수집 데이터 형식 변경

* 수집 데이터를 XML(혹은 JSON) 형식으로 변경해야한다.
* nlohmann/json을 사용하여 수집한 데이터를 json 형식으로 바꿔 파일로 만들어 낸다. 여러가지 라이브러리들 중 nlohmann/json 라이브러리를 선택한 이유는 아래와 같다
  + 사용하기가 간단하다.
  + 전반적으로 성능이 우수하며 보편적으로 사용이 된다.

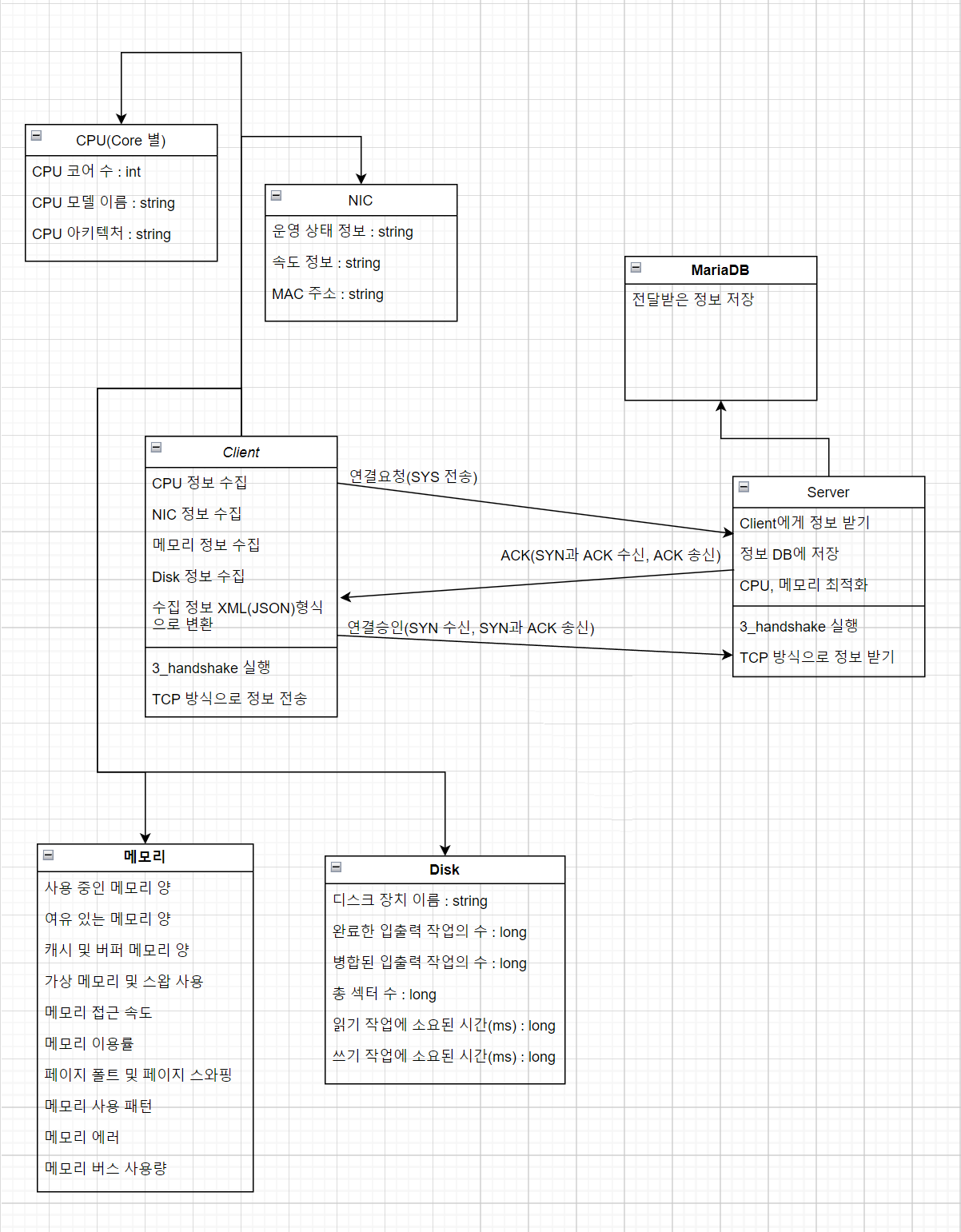
## 5. 데이터 MariaDB 저장

* **$ mysql -u root –p** // DB에 접속(root 계정)
* **select host, user, password from mysql.user;** // 사용자 조회
* **create database DB\_name;** // 새 데이터베이스 생성
* **show databases;** // 데이터베이스 조회
* **drop database DB\_name;** // 데이터베이스 삭제
* **show variables like 'char%';** // 기본 설정 character 확인
* **show create database DB\_name;** // 만들어 놓은 데이터베이스의 character 상태 확인
* **use DB\_name;** // 데이터베이스 사용
* **create table Table\_name ( 테이블 정보 );** // 데이블 생성
* **show tables;** // 사용 중인 데이터베이스 내의 테이블 목록 확인
* **describe Table\_name;** // 테이블 상세 표시
* **show create table Table\_name;** // 테이블 생성문 확인
* **add table Table\_name add [추가할 column명] [타입];** // 테이블 내의 column 추가
* **alter table [테이블 명] change [기존 필드명] [수정할 필드명] [수정할 타입];** // 필드명 수정
* **alter table [테이블 명] [명령어] modify [수정할 타입]** // 테이블 내의 column 수정
* **alter table [테이블 명] drop [필드명];** // 테이블 내의 column 삭제
* **INSERT INTO DB\_name (column1, column2, ...) VALUES (value1, value2, ...)** // 테이블에 데이터 삽입 (column = 열, value = 행)
* **select \* from DB\_name;** // 데이터 조회
* **update DB\_name set column1 = new\_value1, column2 = new\_value2 WHERE condition;** // 기존 데이터 업데이트
* **delete from DB\_name where column1 = [삭제할 데이터];** // 테이블에서 데이터 삭제
* 참고 블로그 : <https://zrr.kr/2BOk>

## 6. 최적화

* 서버 과부하  
  서버부하는 크게 네트워크 부하와 서버의 성능 부하로 분류된다.  
  네트워크 부하는 서버에 접속하는 사람이 폭발적으로 증가하여 트래픽이 급증한 경우의 네트워크에 걸리는 부하이다. 이러한 경우 서버 앞단에서 트래픽 자체를 분산시켜주는 로드밸런서를 사용하거나 서버의 개수를 더 늘리는 방식으로 해결할 수 있다.  
  단일 서버에 걸리는 부하 원인은 크게 2가지로 분류된다. CPU 부하와 I/O 부하이다.
  + CPU 부하  
    CPU 부하가 높은 경우는 서버에서 실행되고 있는 프로그램 자체의 연산량이 많은 경우나 프로그램에 오류등이 발생하는 경우다. 이런 경우에는 프로그램 오류를 제거하거나 알고리즘의 시간, 공간 복잡도를 개선하여 대응해야 한다.
  + I/O 부하  
    I/O 부하가 높은 경우는 서버에서 실행되고 있는 프로그램의 입출력이 많거나, DB나 하드디스크 등의 저장장치로의 접근이 많아 스왑이 발생하는 경우가 대부분이다. 파일 DB, 파일세션, 파일 캐시 등을 사용하면 부하가 더욱 높아진다. 이런 경우에는 특정 프로세스가 극단적으로 메모리를 소비하고 있는지 확인한 후, 프로그램 자체에 오류가 있다면 프로그램을 개선하거나 탑재된 메모리의 용량 자체가 부족한 경우 램을 추가하여 메모리를 증설하는 방법으로 대응할 수 있다. 해당 방법으로 대응할 수 없다면 데이터 자체를 분산(샤딩 or 파티셔닝)하거나 캐시서버 등을 도입하는 방안을 고려할 수 있다.
* DB 과부하  
  DB의 과부하를 잡기 위해 서버를 이중화하여 부하를 분산하도록 하는 클러스터링(Clustering)과 리플리케이션(Reflication) 기술이 있다.  
  클러스터링에는 Active-Active와 Active-StandBy 클러스터링이 있다.
  + Active-Active 클러스터링  
      
    동일한 DB 서버 두 대를 묶고 Active-Active 상태로 운영하면, 하나의 DB 서버가 죽더라도 나머지 DB 서버가 살아있기 때문에 서비스 전체가 중단되는 사태를 방지할 수 있다는 장점을 가지고 있다.  
    하지만 하나의 DB 스토리지를 두 개의 DB 서버가 공유하기 때문에, 병목 현상이 있을 수 있다는 점과 두 개의 DB 서버가 동시에 활성화 되어있는만큼 이전보다 많은 비용이 발생할 수 있다는 단점이 있다.
  + Active-Stand By 클러스터링  
      
    위의 Active-Active 클러스터링의 단점을 보완한 것으로 하나의 Active 서버만을 사용하다가 DB 서버에 문제가 생겼을 때 Fail Over를 통해 Stand By 서버가 Active가 되고, 기존 Active는 Stand By 상태로 전환함으로써 장애에 대응할 수 있다는 장점이 있다.  
    하지만 Fail Over가 바로 일어나는 것이 아니고, 상태가 전환이 되는데 수 초~수 분의 시간이 걸리기 때문에 이 시간 동안 영업 손실이 반드시 생기게 된다는 단점이 있다.
  + 서버만 분산시키는 것이 아니라 DB 스토리지 또한 분산시켜야한다. 이를 위해 파생된 개념이 리플리케이션(Replication)이다. 리플리케이션은 복제를 뜻한다. 2대 이상의 DBMS를 나누어서 데이터를 저장하는 방식을 말하며, 최소 구성은 Master/Slave 구성이다. 기본적인 구조는 아래와 같다.  
      
    Master에 CRUD 동작이 일어나면, 해당 변경사항을 Slave에 반영하는 구조이다. 그러나 이런 구성은 Slave는 단순히 Master의 변경사항을 받아 복제만 하기 때문에, Slave DB 서버가 논다는 단점이 있다. 위와 같은 구조의 경우 Master DB에만 부하가 쏠리게 되어 문제가 생길 가능성이 높아진다.  
      
    그래서 일반적으로 위 사진과 같은 구조로 리플리케이션 구조를 갖춘다. 이 구조에서는 Master 서버와 Salve 서버의 역할을 아래와 같다
  + Master DBMS의 역할  
    웹 서버로부터 등록/수정/삭제 (CUD)의 요청을 담당하게 된다. 이렇게 받은 요청에 대한 처리를 한 후, 즉시 Slave DB 서버에 변경된 데이터를 전달한다.
  + Slave DBMS의 역할  
    Master DBMS로부터 받은 데이터를 반영하고, 주고 읽기 (R)에 대한 요청을 담당하게 된다. 서비스에 맞게 Slave DB 서버를 여러 개 만들 수도 있다.
  + 위와 같이 리플리케이션 사용하면 CUD에 대한 요청과 R에 대한 요청들을 Slave DB 서버들에 분산시킬 수 있으므로 성능 향상을 기대할 수 있다.  
    하지만 각각의 서로 다른 서버로 운영하다 보니 버전 차이를 관리해주어야 한다. 또한 데이터를 동기화 할 때 비동기 방식으로 진행하기 때문에, 일관성 있는 데이터를 얻지 못할 수 있다.

# ◎ 설계서 및 설계도면



# ◎ 비동기 구현

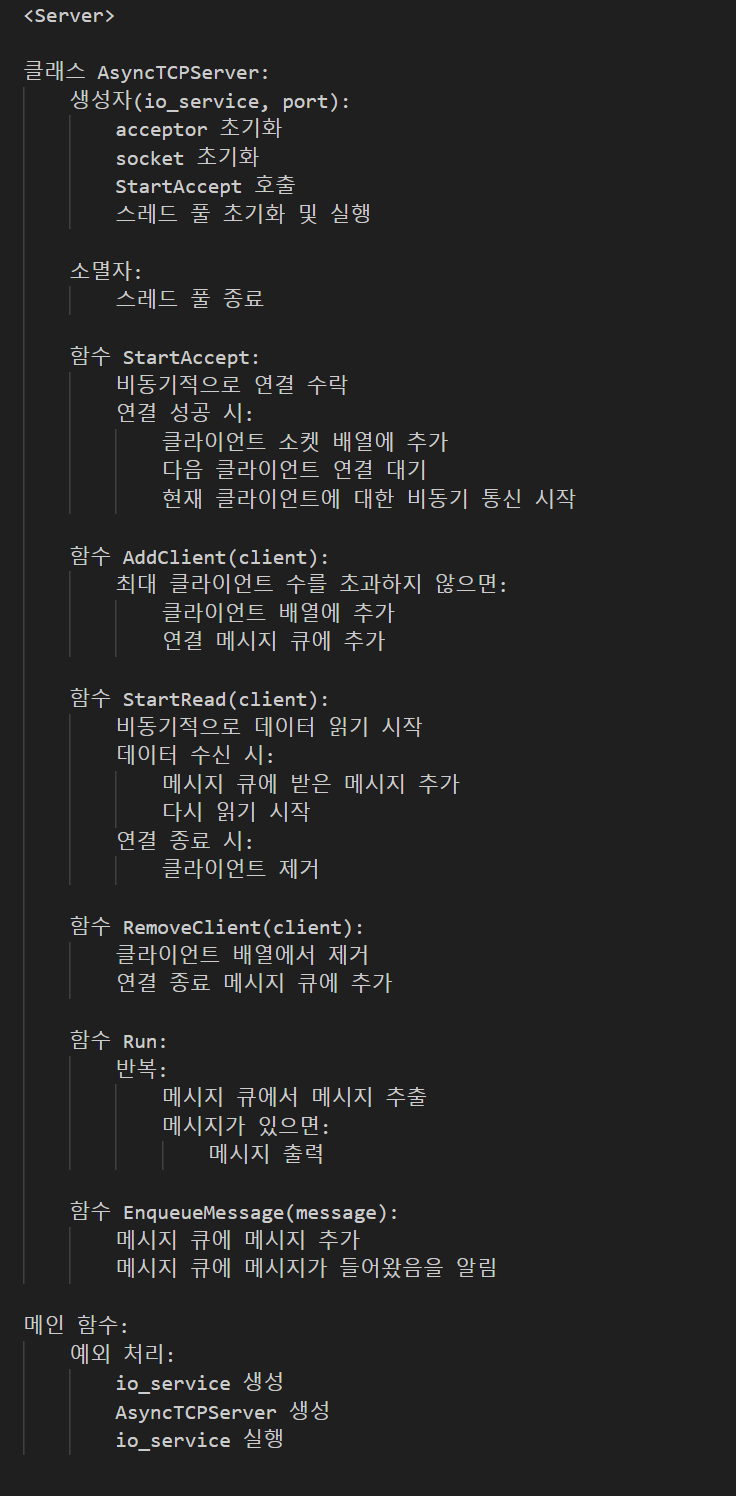
* 비동기(Asynchronous)란 요청을 보냈을 때 응답 상태와 상관없이 다음 동작을 수행하는 것이다.
* C++에서는 표준 라이브러리인 std::future, std::promise, std::packaged\_task, std::async, std::thread 등의 클래스를 통해 비동기 처리를 수행할 수 있다.
  + std::future : 어떤 작업의 결과를 담는 컨테이너. 비동기 작업을 수행하고, 그 결과를 담아 두었다가 나중에 필요할 때 사용할 수 있도록 한다.
  + std::promise : std::future와 함께 사용되는 클래스. 값을 생성하는 쪽에서 사용한다.
  + std::packaged\_task : 함수나 콜러블 객체를 받아 그 결과를 std::future에 저장하는 역할. 함수의 실행을 나중으로 미룰 수 있다.
  + std::async : 비동기 작업을 위한 함수를 제공. 새로운 스레드를 생성하여 함수나 함수 객체를 비동기적으로 실행할 수 있다.
  + std::thread : 함수 객체, 함수 포인터, 람다식과 id를 제공하면 자동으로 thread를 제작해주며 join함수로 blocking 할 수 있다.
* 외부 라이브러리인 Boost.Asio를 통해 구현할 수 있다.
* 각 특징은 아래와 같다.
  + Boost.Asio 특징
    - Boost.Asio는 Boost 라이브러리 컬렉션의 일부로, C++ 표준 라이브러리의 확장으로서 제공된다. C++ 라이브러리의 일부가 아니며 별도로 설치되어야 한다.
    - Boost.Asio는 비동기 I/O 작업을 처리하기 위한 네트워크 프로그래밍, 비동기 파일 I/O, 타이머, 버퍼 조작 등 다양한 기능을 제공한다.
    - Boost.Asio는 C++ 표준의 비동기 기능과 섞이는 것이 일반적이지 않다.
    - Boost.Asio는 특히 네트워크 프로그래밍과 같은 고급 비동기 작업에 뛰어난 성능과 기능을 제공한다.
  + 표준 라이브러리 특징
    - 별도의 외부 라이브러리를 설치할 필요 없이 사용할 수 있다.
    - Boost.Asio보다는 간단한 비동기 작업을 위한 것으로 설계되었다. 주로 std::async와 std::future를 중심으로 하며, 네트워크나 저수준 I/O 작업에 특화되지 않았다.
    - 표준 라이브러리에 통합되어 있으므로, 표준 라이브러리의 다른 부분과 자연스럽게 통합된다.
    - 표준 라이브러리 기능은 간단한 비동기 작업에 대한 표준화된 접근을 제공한다..

# ◎ 예상 결과

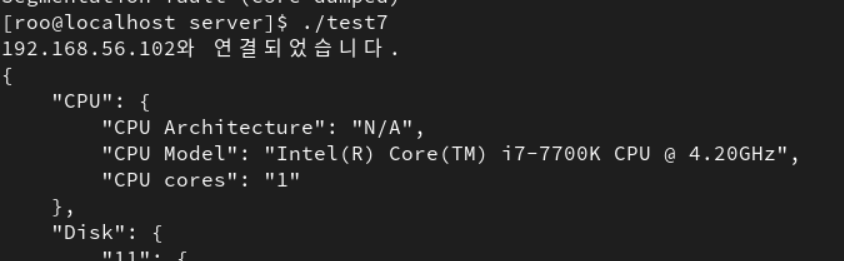
1. Client가 CPU, NIC, 메모리, Disk의 정보를 수집한다.
2. Server는 비동기 상태로 상시 유지된다.
3. Client가 Server에 접속을 시도한다.
4. 접속에 성공한다면 Client가 Server에 접속할 정보를 수집하여 XML(혹은 JSON) 형식으로 변환시켜 파일로 생성한다
5. Client가 Server에게 XML(혹은 JSON) 형식 파일 내용 전송한다.
6. Server가 수신한 정보를 DB에 저장한다.

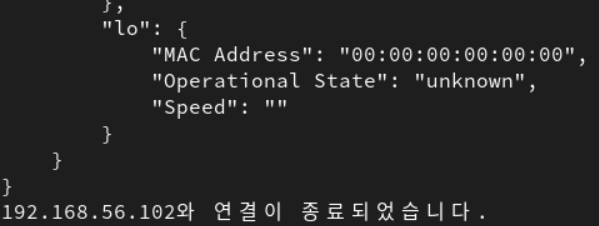
# ◎ 각종 샘플

## Server 설계



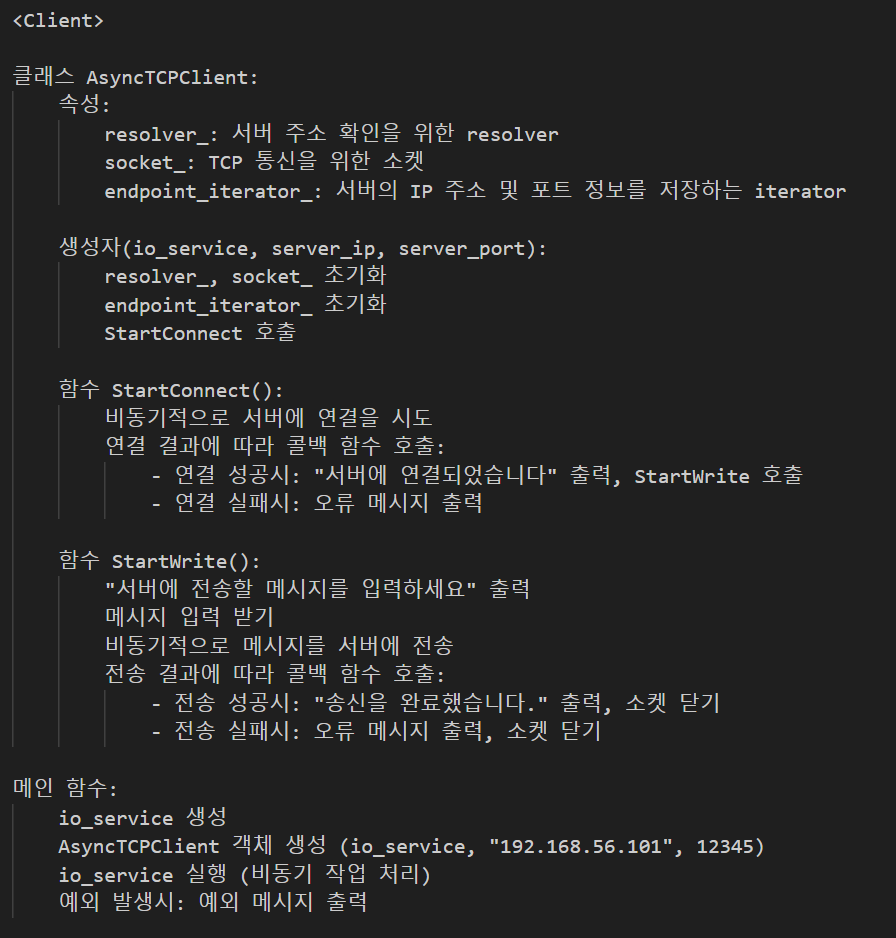
Server 샘플 코드 결과



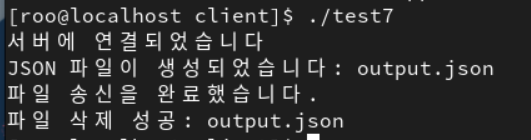


서버가 실행되면 상시로 대기하다 클라이언트가 연결이 된다면 연결된 클라이언트의 IP를 띄우며 연결되었다고 표시한다. 이후 받은 데이터들을 출력한 후(출력 부분 삭제 예정) DB에 저장시킨다. 클라이언트는 정보만 전송한 후 연결이 해제되기 때문에 해제된 클라이언트의 IP 주소를 띄우며 종료되었다고 출력시킨다.

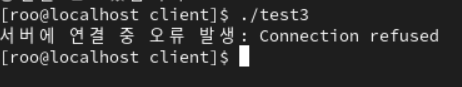
## Client 설계



Client 샘플 코드 결과

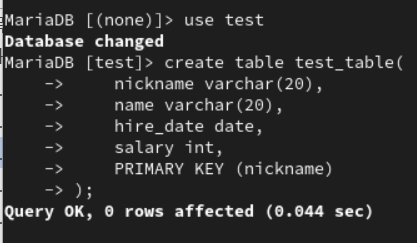
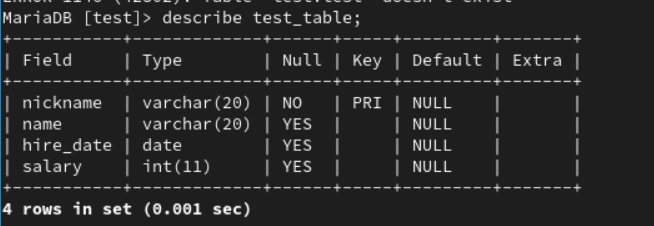
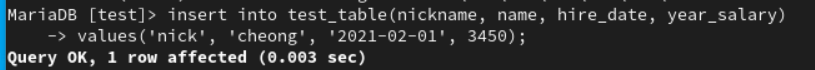
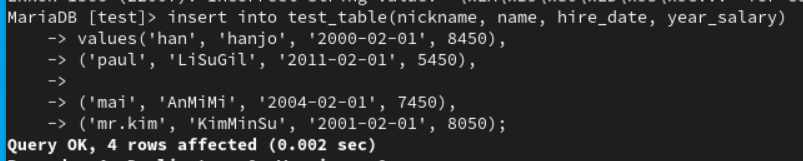
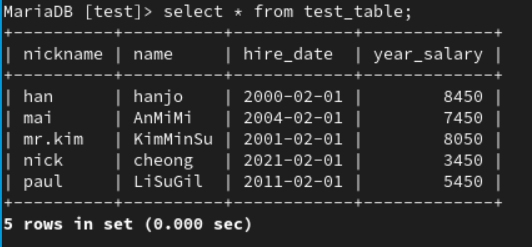
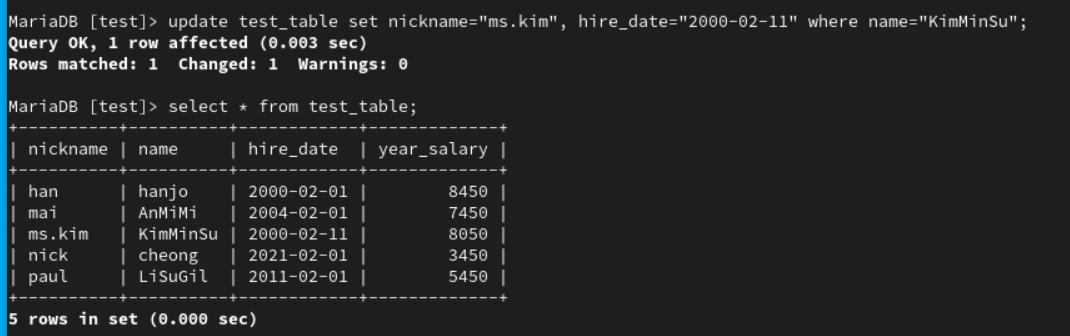
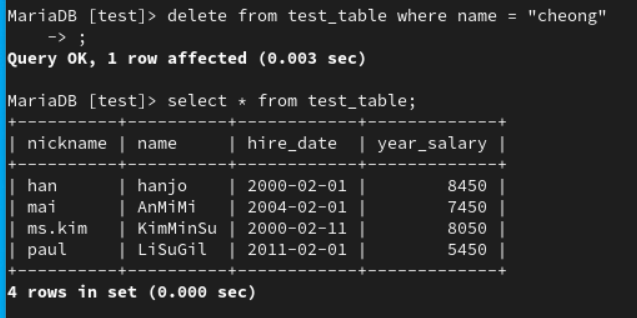


실행하면 Server와 연결 시도 후 연결 성공 시 연결되었다고 출력된다. 이후 CPU, NIC, Memory, Disk의 정보들을 수집하여 josn 형식으로 만들어 output.json 이라는 파일을 생성한다. 생성에 성공하고 파일이 정상적으로 열린다면 내용을 읽어 Server에 송신하고 송신이 완료된다면 json 파일이 삭제된다.

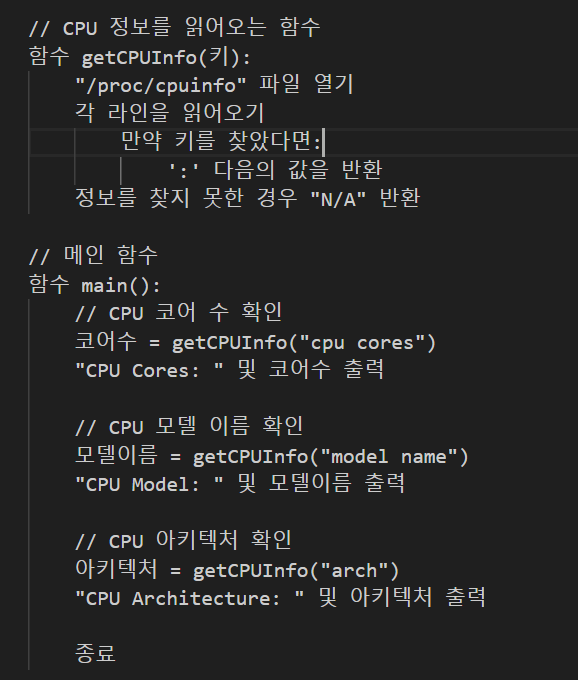


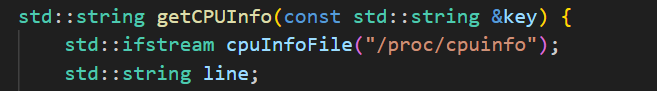
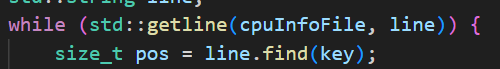
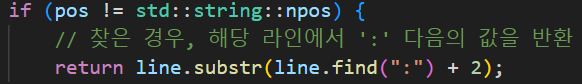
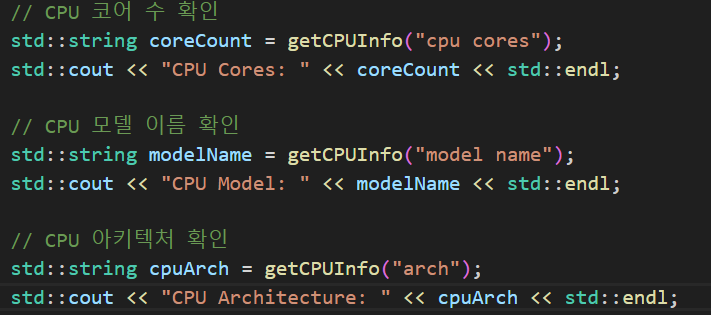
만일 Server와 연결이 실패하였다면, 위와 같은 오류 메시지를 출력하고 종료된다.

## MariaDB 사용

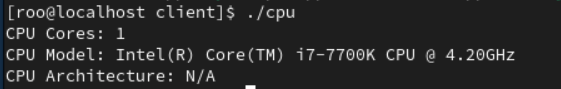
* DB에 테이블 생성 샘플  
    
  생성한 test 라는 DB에 들어가, test\_table 이란 테이블을 내용을 포함해 만든다.
* 테이블 상세 표시 확인 샘플  
    
  describe [테이블명] 을 사용해 상세 내용을 확인할 수 있다.
* 데이터를 단일, 혹은 묶어 테이블 저장 샘플  
   
* 저장된 데이터 출력 샘플  
  
* 테이블 수정 샘플 // KimMinSu의 nickname과 dire\_date의 정보를 바꿈  
  
* 테이블 삭제 샘플 // cheong 삭제  
  

## CPU 설계



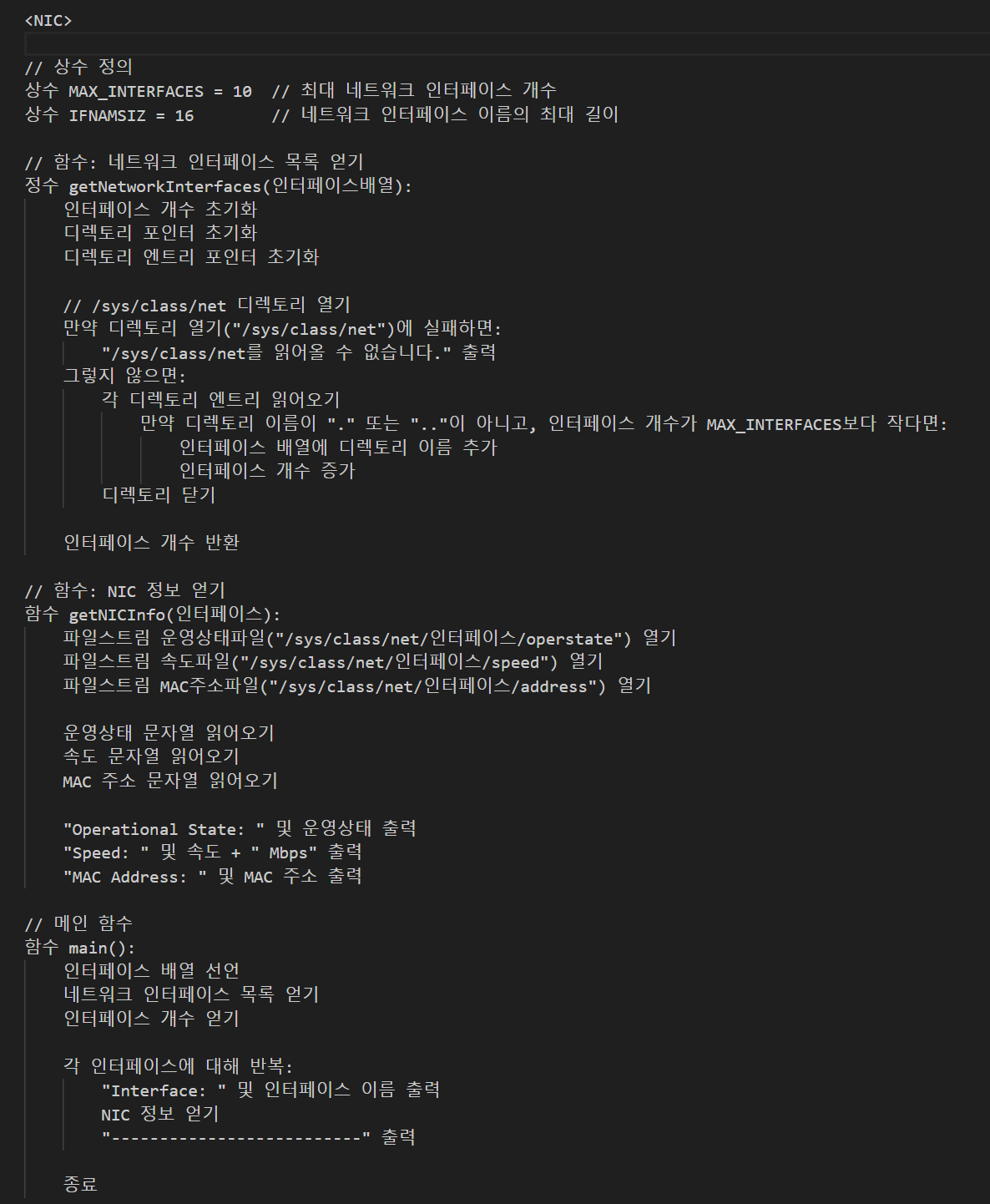
* /proc/cpuinfo 파일을 열기 위해 fstream 클래스를 사용한다.
* 컴퓨터의 proc/cpuinfo 파일을 읽어와 특정 CPU 정보를 추출한다.  
  
  + “/proc/cpuinfo” 파일을 열은 후 한 줄씩 읽어올 문자열을 선언한다.
* 반복문을 파일을 처음부터 끝까지 탐색하여 key와 동일한 문자열이 있는지 파악한다. 만약 동일한 문자열이 없다면 std::string::npos를 반환한다.  
  
  + getline(cpuInfoFile, line) : cpuInfoFile명의 파일을 한 줄씩 읽어 line에 저장한다..
  + line.find(key) : line에 저장된 값이 key의 값과 같은지 확인한다.
* key와 동일한 값을 찾는다면 ‘ : ‘ 다음에 입력되어 있는 값을 추출하여 반환시킨다.  
  
  + line.substr(line.find(“:”) + 2) : 찾아낸 값의 ‘ : ‘ 다음 부분의 값을 반환시킨다.
* 존재하지 않는다면 “N/A” 값을 반환시킨다.  
  
* cpu cores, model name, arch 에 대한 정보를 출력한다. (출력 부분을 이제 XML과 JSON파일로 만들어야 한다.)  
  

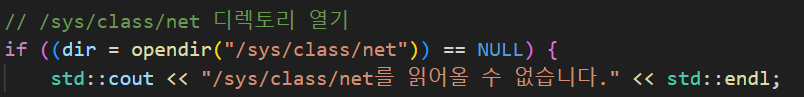
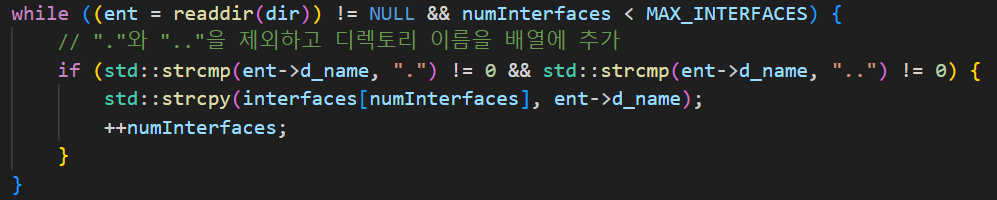
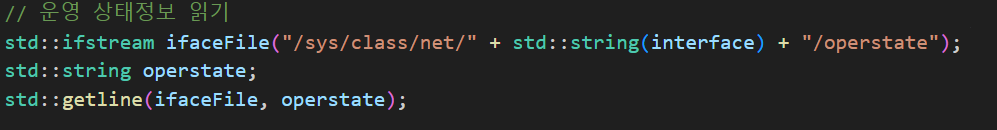
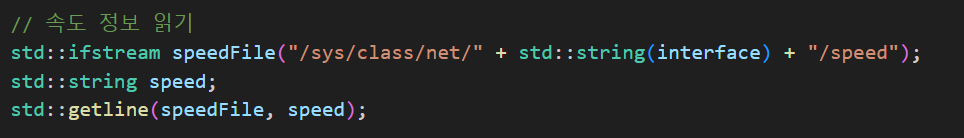
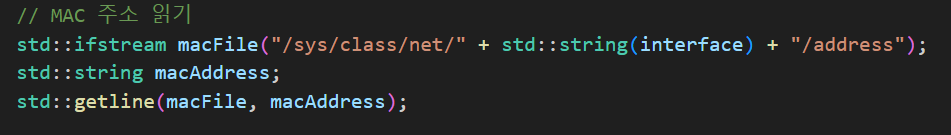
CPU 관련 결과



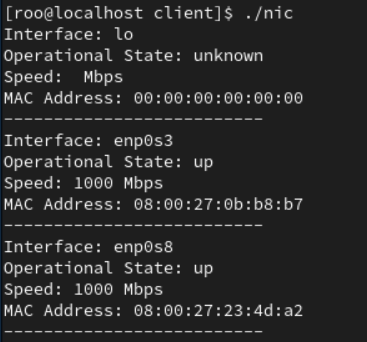
다양한 CPU에 대한 정보 중 CPU의 코어 개수, 모델명, 아키텍처를 수집한다.

## NIC 설계



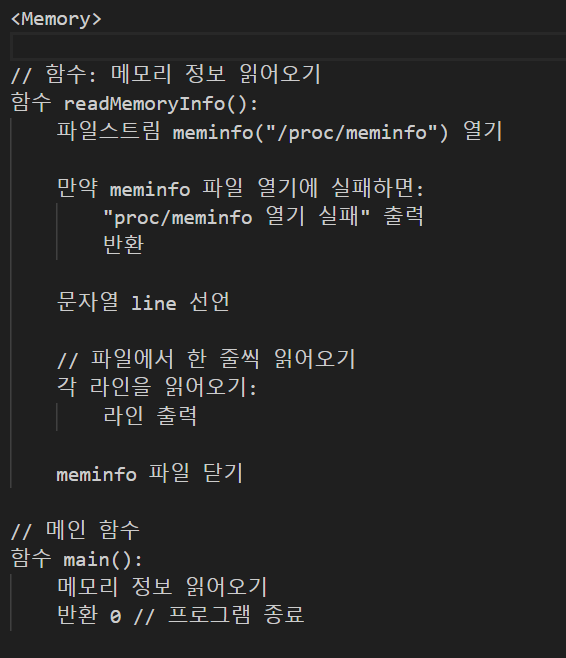
* /sys/class/net 파일을 읽기 위해 fstream 클래스를 사용한다.
* /sys/class/net 파일을 열어 읽어온다. 만약 비어있을 경우 오류 메시지를 출력한다.  
  
* 디렉토리의 끝까지 항목을 읽을 때까지 혹은 numInterfaces < MAX\_INTERFACES가 되기 전까지 반복한하며 디렉토리의 항목을 처리한다.  
  
  + std::strcmp(ent->d\_name, “ “) : 현재와 상위 디렉토리를 제외한 나머지 디렉토리를 처리한다.
  + std::strcpy() : interfaces[numInterfaces]는 배열의 다음 사용 가능한 위치에 해당하며, 현재 디렉토리의 이름이 복사된다.
  + 반복문을 빠져나가면 closedir(dir); 를 사용해 dir를 닫아준 후 numInterfaces를 반환시킨다.
* 운영 상태 정보를 읽은 후 출력한다.  
  
  + std::ifstream을 이용하여 ifaceFile 파일을 생성한다. 파일 경로는 "/sys/class/net/" + std::string(interface) + "/operstate"로 지정된다.
  + std::string operstate; : 파일에서 읽어온 문자열을 저장할 문자열이다.
  + std::getline(ifaceFile, operstate); : ifaceFile에서 읽어온 정보를 operstate에 저장한다.
  + operstate 정보를 출력한다.
* 속도 정보를 읽은 후 출력한다.  
  
* MAC 주소를 읽은 후 출력한다.  
  

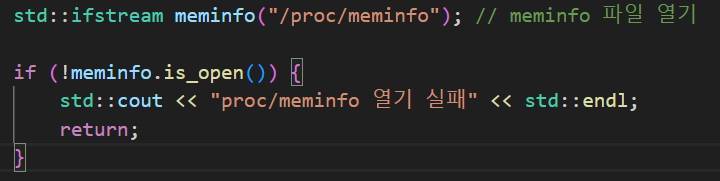
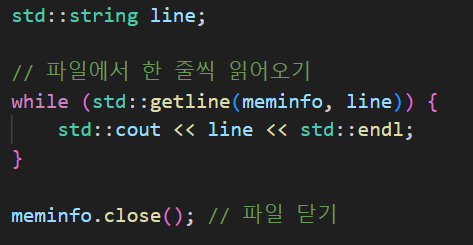
NIC 관련 결과



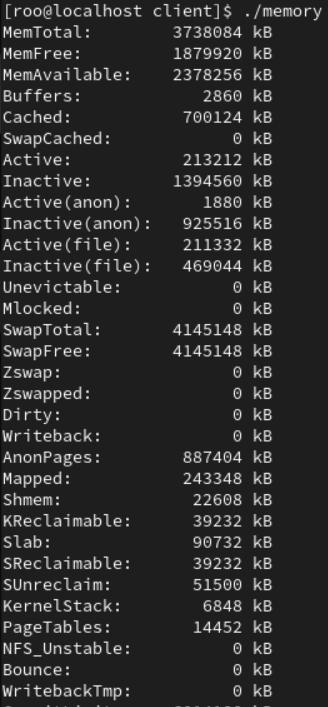
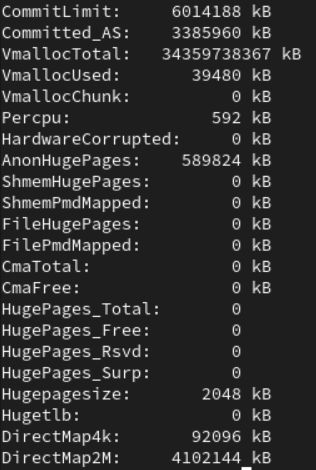
각 nic의 인터페이스, 운영 상태, 속도, MAC 주소 정보를 수집한다.

## 메모리 설계



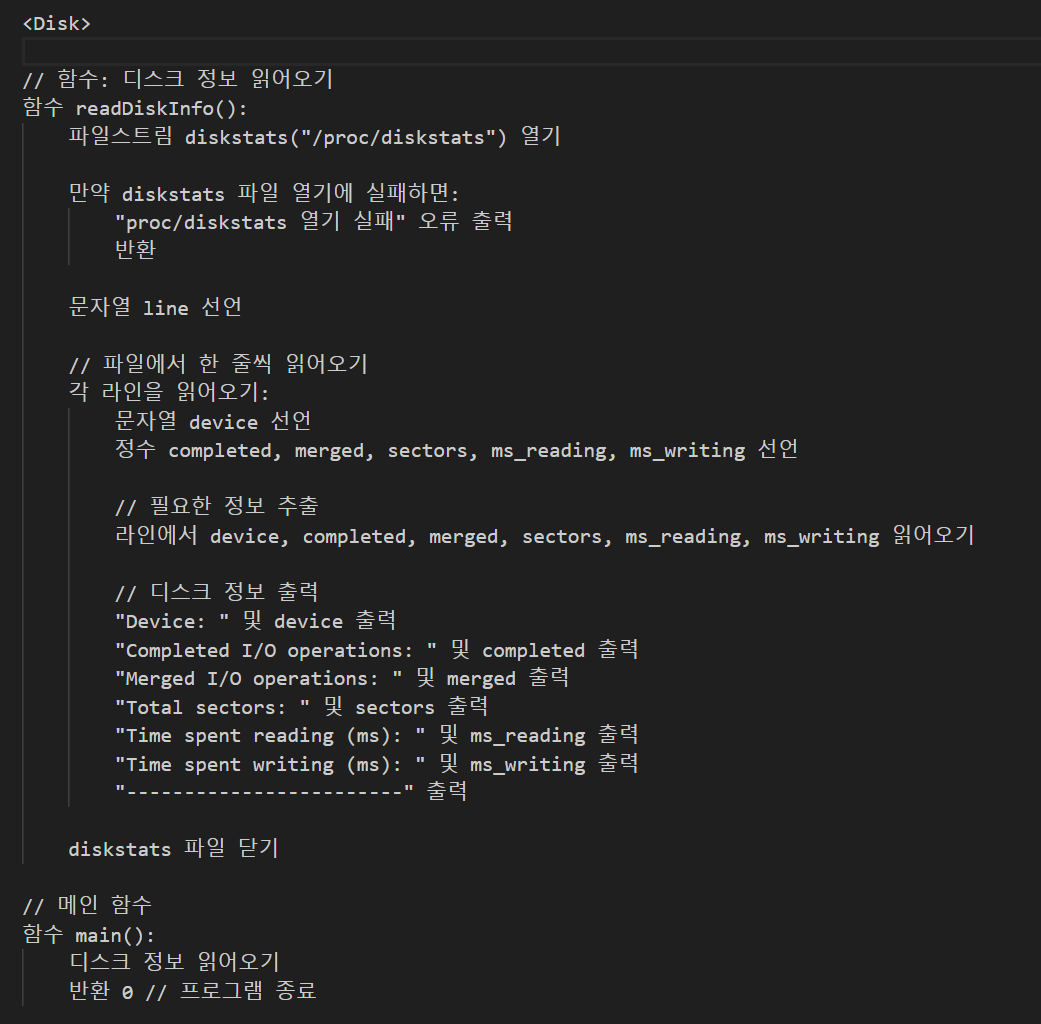
* /proc/meminfo 파일을 읽어내기 위해 fstream 클래스를 사용한다.
* proc/meminfo 파일을 연다. 만약 실패 시 오류 메시지를 출력한다.  
  
* proc/meminfo 파일을 한 줄씩 읽는다. 읽을 때마다 한 줄씩 출력을 하고 다 읽는다면 반복문을 빠져나간 후 파일을 닫는다.  
  

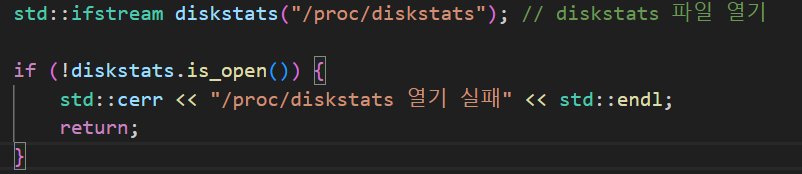
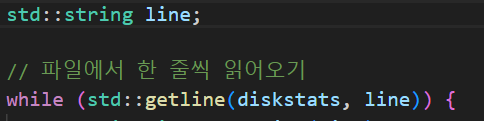
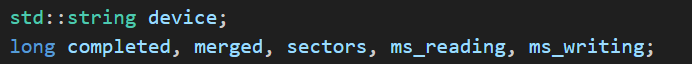
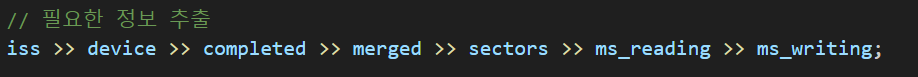
메모리 관련 결과

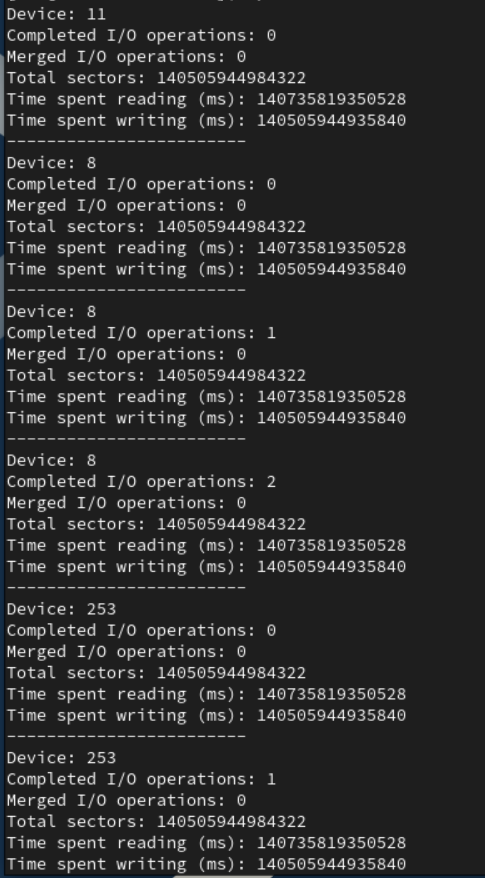
모든 메모리의 정보를 수집한다.

## Disk 설계



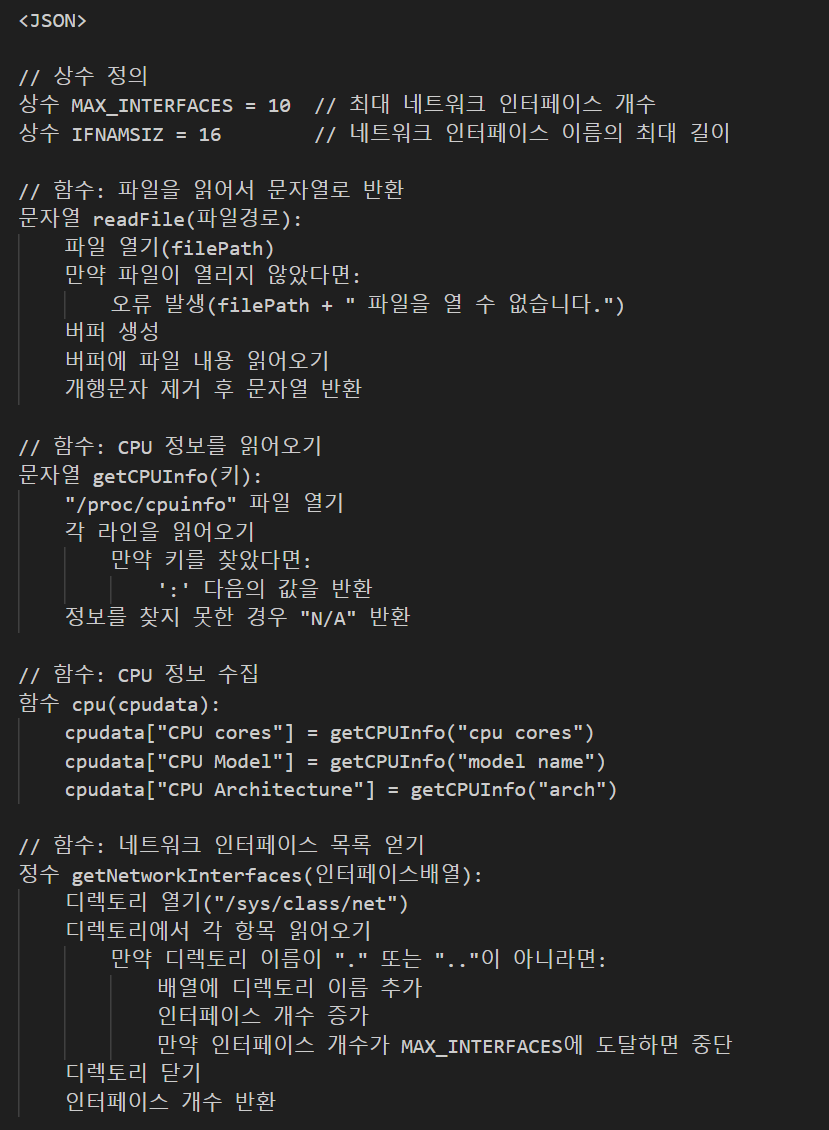
* /proc/diskstats 파일을 읽어내기 위해 fstream 클래스를 사용한다.
* /proc/diskstats 파일을 연다. 만약 실패 시 오류 메시지를 출력한다.  
  
* getline을 사용해 파일을 한 줄씩 읽어오며 반복한다.  
  
* std::istringstream 클래스를 사용하여 line 문자열을 스트림으로 변환시켜 문자열에서 공백을 기준으로 분리된 각각의 데이터를 추출한다.  
  
* 이후 std::string을 이용한 device와 long형식인 completed, merged, sectors, ms\_reading, ms\_writing 변수들을 선언한다.  
  
* >> 를 이용해 스트림에서 값을 추출하여 변수에 할당한다.  
  
* Device, Completed I/O operations, Merged I/O operations, Total sectors, Time spent reading (ms), Time spent writing (ms) 정보를 출력한다.

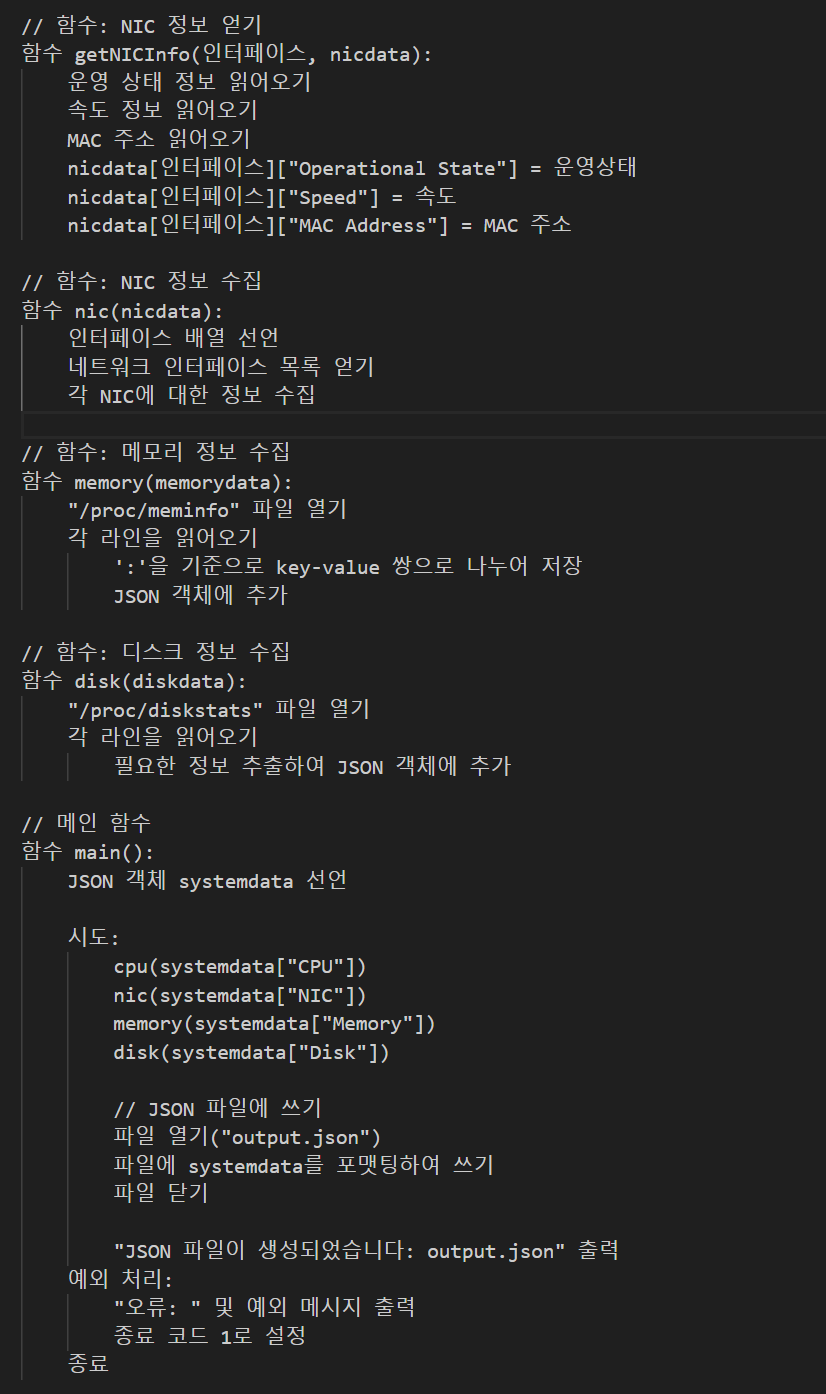
Disk 관련 결과



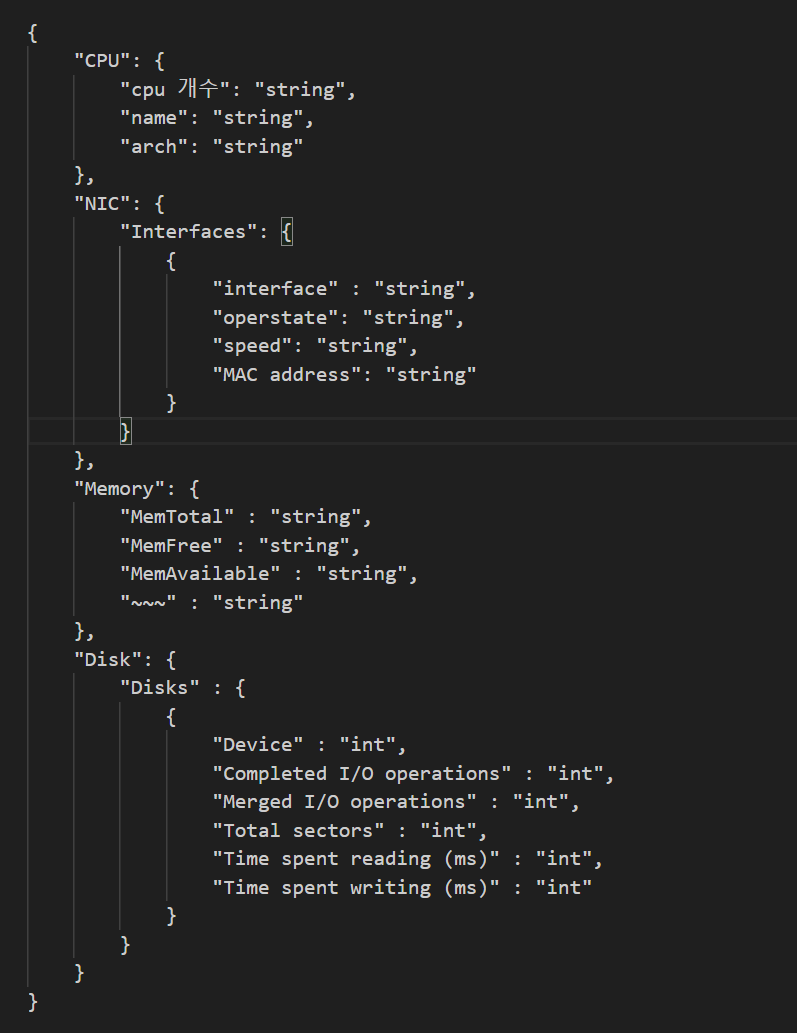
디스크의 장치 이름, 입출력 작업의수, 병합된 입출력 작업의 수, 총 섹터 수, 읽기 작업에 소요된 시간, 쓰기 작업에 소요된 시간 을 수집한다

## Json 설계





CPU, NIC, Memory, Disk의 내용을 읽어낸 후 nlohmann/json을 이용하여 JSON 파일로 만들어 낸다.



CPU, NIC, Memory의 경우에는 모든 내용을 string형식으로, Dkisk의 경우에는 int형식으로 받으며JSON에 작성한다. 아래는 만들어진 JSON의 결과이다.

