Zadaniem jest serwer pod Linuxa, który będzie:
- obsługiwał dowolną ilość równoległych połączeń (ograniczonych zasobami systemu),
- odpowiadał możliwie najszybciej na wysyłane zapytania (powinny być obsługiwane równolegle),
- przygotowany na problemy z transmisją,
- utrzymywał przychodzące połączenia; zamknięcie powinno nastąpić tylko po zakończeniu połączenia przez klienta lub przy braku jego aktywności przez 30s.

- zapisywał przesłane informacje w bazie danych A (dowolna konstrukcja tabel)
- odczytywał informacje z bazy danych A
- zapisywał logi zapisu informacji (np. zmieniono wartoś X na Y) w odrębnej bazie B (dowolna konstrukcja tabel)

Zapytania będą oparte na prostym protokole opakowanym w JSON:
{ "cmd": <POLECENIE>, "args": \{ ... }

}
w odpowiedzi:
{ "status": "ok"/"error, ... }

Polecenia do obsługi:
STAT - serwer odsyła statystyki (w dowolnej postaci):
- ilość czasu od uruchomienia serwer (w sekundach)
- ilość czasu od nawiązania aktualnego połączenia (w sekundach)
- ilość łącznie odebranych zapytań
- ilość aktualnie utrzymywanych połączeń,

INC <liczba typu "long long"> - serwer zwiększa ilość wystąpień podanej liczby w swojej strukturze, będącej wspólną dla całego serwera (nie tylko dla aktualnego połączenia); w odpowiedzi zwracany jest aktualny licznik dla podanej liczby,
{ "cmd": "INC", "args": { "number": 49 }} -> { "status": "ok", "hits": 4 }

GET <liczba typu "long long"> - zwraca ilość wystąpień podanej liczby,
{ "cmd": "GET", "args": { "number": 49 }} -> { "status": "ok", "hits": 4 }

SLEEP <czas oczekiwania (int)> - serwer czeka podaną ilość sekund, po czym odpowiada OK,
{ "cmd": "SLEEP", "args": { "delay": 10 }} -> { "status": "ok" }

END - serwer odpowiada OK i kończy połączenie.
{ "cmd": "END" }

-> { "status": "ok" }
... i rozłączenie

WRITE <klucz typu "string"> <wartość> - zapisuje do bazy wartość w danym kluczy (lub zmienia istniejący)
{ "cmd": "WRITE", "args": { "key": "test key", "value": "test value" }} -> { "status": "ok" }

READ <klucz> - odsyła wartość klucza zapisaną w bazie (lub łańcuch pusty gdy nie ma)
{ "cmd": "READ", "args": { "key": "test key" }} -> { "status": "ok", "value": "test value" }

DEL <klucz> - usuwa wartość klucza zapisaną w bazie
{ "cmd": "DEL", "args": { "key": "test key" }} -> { "status": "ok" }

Jeżeli serwer otrzyma nieznane polecenie, powinień odesłać informację o błędzie:,
np. { status: "error" }

Serwer musi być przygotowany na równoległy dostęp (odczyt i zapis) do danych w kilku połączeniach.
Implementacja C++ z użyciem biblioteki Boost (http://www.boost.org/).
Baza danych: mysql.
JSON z użyciem jsoncpp (https://github.com/open-source-parsers/jsoncpp).