Synchronicznie czy asynchronicznie? Oto jest pytanie!

(quiz z odpowiedzią na końcu)

Bartek ’BaSz’ Szurgot

bartek.szurgot@baszerr.eu

2015-02-12

Na początku był kod...

- Algorytmy zmieniające oblicze świata:

```
1  #include <cmath>
2  #include "Logger.hpp"
3
4  void threadCall(unsigned id, unsigned n)
5  {
6    for(unsigned x=0; x<n; ++x)
7    {
8      MYLOG((x*100)/n << "%␣done" );
9      const auto y = cos( sin(x) );
10     MYLOG(id << ":␣cos(sin(" << x << "))=" << y);
11   }
12 }
```

- Z czym może być problem?

Logger

- Logger – aka globalna zmienna dzielona

Logger

- Logger – aka globalna zmienna dzielona
- Logi wpływają na czas

Logger

- Logger – aka globalna zmienna dzielona
- Logi wpływają na czas
- Wątki a czas
  - Niejawna synchronizacja na logach
  - Wyścigi znikają jak włącza się logi debugowe

Logger

- Logger – aka globalna zmienna dzielona
- Logi wpływają na czas
- Wątki a czas
  - Niejawna synchronizacja na logach
  - Wyścigi znikają jak włącza się logi debugowe
- Formatowanie podczas trzymania blokady...
  - *MYLOG("the answer is: " « deepThought())*
  - To może chwilę zająć...

Logger

- Logger – aka globalna zmienna dzielona
- Logi wpływają na czas
- Wątki a czas
  - Niejawna synchronizacja na logach
  - Wyścigi znikają jak włącza się logi debugowe
- Formatowanie podczas trzymania blokady...
  - *MYLOG("the answer is: " « deepThought())*
  - To może chwilę zająć...
- Jak wybrać najlepsze podejście?

Logger

- Logger – aka globalna zmienna dzielona
- Logi wpływają na czas
- Wątki a czas
  - Niejawna synchronizacja na logach
  - Wyścigi znikają jak włącza się logi debugowe
- Formatowanie podczas trzymania blokady...
  - *MYLOG("the answer is: " « deepThought())*
  - To może chwilę zająć...
- Jak wybrać najlepsze podejście?
- Zmierzyć!
- Jak wygląda implementacja?

Prosty logger

```
1  extern std::mutex    g_logMutex;
2  extern std::ofstream g_logFile;
3
4  #define MYLOG(msg) do { \
5      std::lock_guard<std::mutex> lock(g_logMutex); \
6      g_logFile << __FILE__ << ":" << __LINE__ \
7                << "␣" << msg << '\n'; \
8    } while(false)
```

Prosty logger

```
1  extern std::mutex    g_logMutex;
2  extern std::ofstream g_logFile;
3
4  #define MYLOG(msg) do { \
5      std::lock_guard<std::mutex> lock(g_logMutex); \
6      g_logFile << __FILE__ << ":" << __LINE__ \
7                << "␣" << msg << '\n'; \
8    } while(false)
```

- *MYLOCK("result: " « heavyStuff())*
- To może potrwać. . .
- Łatwiej o zakleszczenie (ang. deadlock)

Formatowanie przed blokowaniem

```
1  extern std::mutex    g_logMutex;
2  extern std::ofstream g_logFile;
3
4  #define MYLOG(msg) do { \
5      std::stringstream ss; \
6      ss << __FILE__ << ":" << __LINE__ \
7                 << "␣" << msg << '\n'; \
8      std::lock_guard<std::mutex> lock(g_logMutex); \
9      g_logFile << ss.str(); \
10   } while(false)
```

Kolejka blokująca

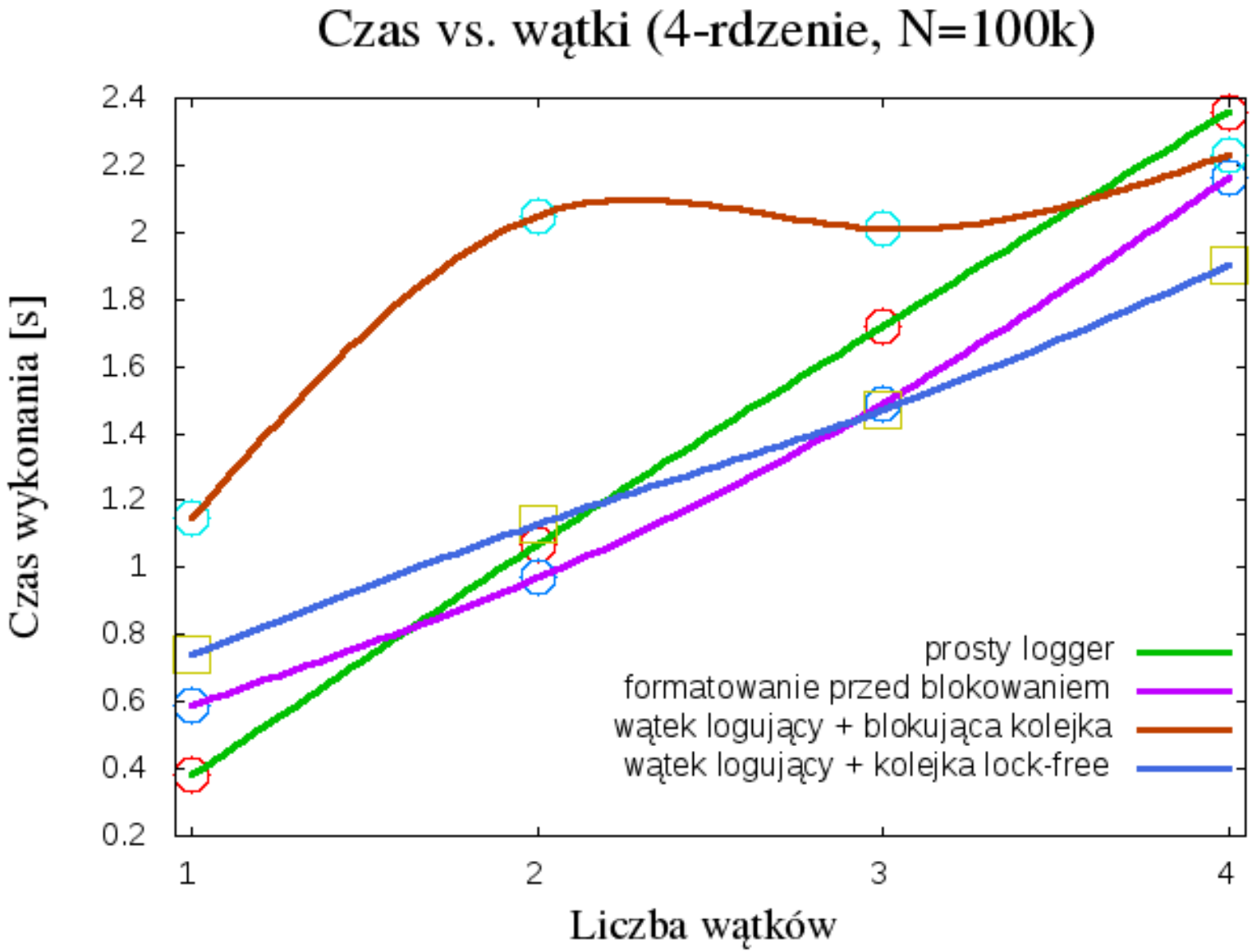
```
1  using Queue = Fifo<std::string>;
2  extern Queue g_queue;
3
4  #define MYLOG(msg) do { \
5      std::stringstream ss; \
6      ss << __FILE__ << ":" << __LINE__ \
7         << "␣" << msg << std::endl; \
8      auto str = ss.str(); \
9      Queue::lock_type lock(g_queue); \
10     g_queue.push( std::move(str) ); \
11     g_queue.notify(); \
12   } while(false)
```

Kolejka lock-free

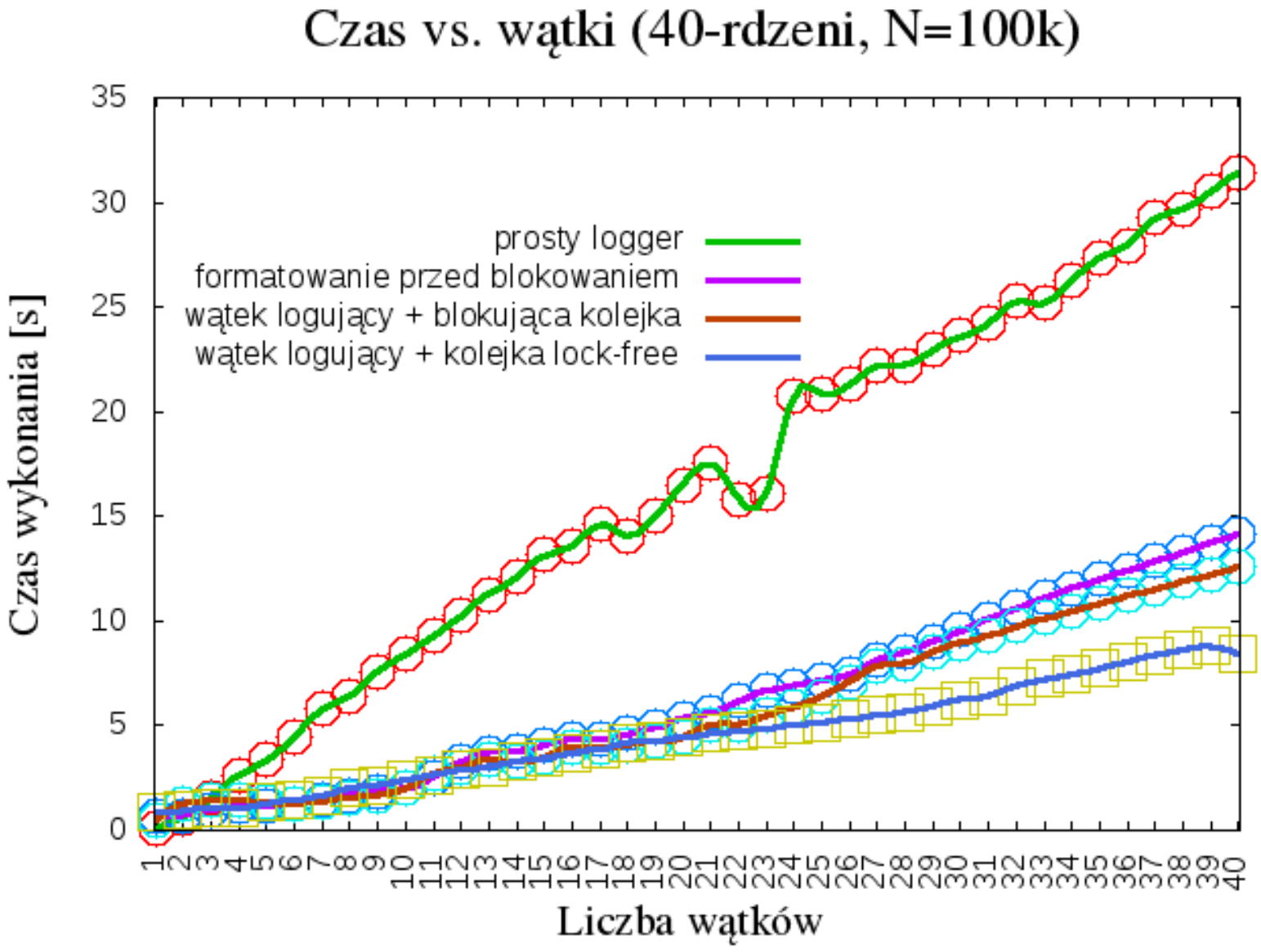
```
1  using Queue = boost::lockfree::queue<std::string*>;
2  extern Queue g_queue;
3
4  #define MYLOG(msg) do { \
5      std::stringstream ss; \
6      ss << __FILE__ << ":" << __LINE__ \
7         << "␣" << msg << std::endl; \
8      auto ptr = std::make_unique<std::string>(ss.str()); \
9      while(not g_queue.push(ptr.get())) { } \
10     ptr.release(); \
11   } while(false)
```

Chwila prawdy

Chwila prawdy



More power!


Czas vs. wątki (40-rdzeni, N=100k)
Czas wykonania [s]
Liczba wątków
prosty logger
formatowanie przed blokowaniem
wątek logujący + blokująca kolejka
wątek logujący + kolejka lock-free
0
5
10
15
20
25
30
35
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

Wyciągnięte wnioski

- Blokujące I/O – nie takie złe. . .

Wyciągnięte wnioski

- Blokujące I/O – nie takie złe...
- Nie wykryto złotego młotka...
  - Proste przypadki – proste rozwiązania!
  - Różne charakterystyki – różne algorytmy
  - Lock-free niekoniecznie najszybsze
  - Lock-free też blokuje
    - Synchronizacja na poziomie sprzętu
    - Alokatory mogą blokować
  - Wyniki pomiarów – często zaskakujące

Wyciągnięte wnioski

- Blokujące I/O – nie takie złe...
- Nie wykryto złotego młotka...
  - Proste przypadki – proste rozwiązania!
  - Różne charakterystyki – różne algorytmy
  - Lock-free niekoniecznie najszybsze
  - Lock-free też blokuje
    - Synchronizacja na poziomie sprzętu
    - Alokatory mogą blokować
  - Wyniki pomiarów – często zaskakujące
- Asynchroniczność nie jest darmowa
- Asynchroniczne I/O dla wolniejszych operacji

Wyciągnięte wnioski

- Blokujące I/O – nie takie złe...
- Nie wykryto złotego młotka...
  - Proste przypadki – proste rozwiązania!
  - Różne charakterystyki – różne algorytmy
  - Lock-free niekoniecznie najszybsze
  - Lock-free też blokuje
    - Synchronizacja na poziomie sprzętu
    - Alokatory mogą blokować
  - Wyniki pomiarów – często zaskakujące
- Asynchroniczność nie jest darmowa
- Asynchroniczne I/O dla wolniejszych operacji
- Mierzenie
  - Zawsze warto
  - Szczególnie "oczywiste oczywistości"
  - Oraz "sytuacje z góry pewne"
  - Nie ma migania się! :P

?