Optymalizacja Kodu Na Różne Architektury
Ćwiczenie 6

1 Wprowadzenie

Rozważmy następującą procedurę usuwającą znaki sterujące ze string-a

Listing 1: Bazowa wersja kodu

```
//requires additional changes to the code to make it work

#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>
#include <string>
#include <iostream>

#define REPEATS 50000

static double gtod_ref_time_sec = 0.0;

/* Adapted from the bl2_clock() routine in the BLIS library */

double dclock()
{
  double the_time, norm_sec;
  struct timeval tv;
  gettimeofday( &tv, NULL );
  if ( gtod_ref_time_sec == 0.0 )
    gtod_ref_time_sec = ( double ) tv.tv_sec;
  norm_sec = ( double ) tv.tv_sec - gtod_ref_time_sec;
  the_time = norm_sec + tv.tv_usec * 1.0e-6;
  return the_time;
```

```
}



std::string remove_ctrl(std::string s){
    std::string result;
    for (int i = 0; i < s.length(); i++){
        if (s[i] >= 0x20)
            result = result + s[i];
    }
    return result;
}


int main( int argc, const char* argv[] )
{
  int i,j,k,iret;
  double dtime;

  std::cout << "call to remove\n";

  std::string s;
  std::string result;

  std::string line;
  while (getline(std::cin, line)){
      s += line + "\n";
  }

  dtime = dclock();
  for (int i = 0; i < REPEATS; i++){
      result = remove_ctrl(s);
  }
  dtime = dclock() - dtime;

  std::cout << result << "\n";
  std::cout << "Time: " << dtime << "\n";
  fflush( stdout );

  return iret;
}
```

2 Ćwiczenia

Przy kolejnych krokach proszę wykonywać kopie aktualnych wersji stanu kodu. Będą potrzebne przy ostatnim punkcie oraz na kolejnych zajęciach. Proszę na bieżąco notować czasy wykonywania się obliczeń.

2.1

Proszę skompilować procedurę bez optymalizacji.

```
maciekw@Banach:~/studenci/OORA/skrypty/lab6> g++ str1.c++
maciekw@Banach:~/studenci/OORA/skrypty/lab6> ./a.out < lab1.tex
Time: 55.6916
maciekw@Banach:~/studenci/OORA/skrypty/lab6> g++ str1.c++ -O2
maciekw@Banach:~/studenci/OORA/skrypty/lab6> ./a.out < lab1.tex
Time: 50.6151
```

Nawet z opcją **-O2** czas wykonania kodu jest nieakceptowalny. Spróbujmy z pomocą profilera, aby znaleźć źródło problemu.

```
maciekw@Banach:~/studenci/OORA/skrypty/lab6> g++ -pg str1.c++
maciekw@Banach:~/studenci/OORA/skrypty/lab6> ./a.out < lab1.tex
Time: 56.8771
maciekw@Banach:~/studenci/OORA/skrypty/lab6> gprof ./a.out
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 81.90      2.29     2.29    50000    45.70    55.52  remove_ctrl(std::__cx
 17.60      2.78     0.49 343700000     0.00     0.00  std::__cxx11::basic_s
  0.72      2.80     0.02      220    91.11    91.11  std::__cxx11::basic_st
  0.00      2.80     0.00        2     0.00     0.00  dclock()
  0.00      2.80     0.00        1     0.00     0.00  _GLOBAL__sub_I__Z6dclo
  0.00      2.80     0.00        1     0.00     0.00  __static_initializatior
```

Możemy zaobserwować, że spędzamy 82% czasu wewnątrz funkcji **remove_ctrl**. Jednak mamy 343700000 wywołań do funkcji **std::allocator**, oraz spędzamy w niej aż 17% czasu. Przy kompilowaniu z opcją **-O2** profiler może nie dać poprawnych wyników (nie będą znane wywołane funkcje).

```
maciekw@Banach:~/studenci/OORA/skrypty/lab6> g++ -pg -O2 str1.c++
maciekw@Banach:~/studenci/OORA/skrypty/lab6> ./a.out < lab1.tex
Time: 56.8771
maciekw@Banach:~/studenci/OORA/skrypty/lab6> gprof ./a.out
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
```

```
 time    seconds    seconds     calls   ns/call   ns/call  name
 64.88       1.30      1.30                                remove_ctrl(std::__cx
 35.43       2.02      0.71 343700000      2.07      2.07  frame_dummy
  0.00       2.02      0.00         1      0.00      0.00  _GLOBAL__sub_I__Z6dcl
  __static_initialization_and_destruction_0(int, int)
```

2.2

Proszę wykonać kopię procedury

```
maciekw@Banach:~/studenci/OORA/skrypty/lab6> cp str1.c++ str2.c++
```

Bardzo duża liczba wywołań do **std::allocator** może wskazywać na ciągłe wywołanie konstruktora.

```
result = result + s[i];
```

W powyższej linijce tworzymy nowego string-a. Dokonajmy pierwszej modyfikacji poprzez modyfikowania string-a wynikowego, zamiast tworzenie nowego w każdym kroku.

```
result += s[i];
```

Proszę skompilować procedurę z opcją, oraz bez opcji **-O2**.
Jaki jest czas działania?
Co pokazuje profiler?

2.3

Proszę wykonać kopię procedury

```
maciekw@Banach:~/studenci/OORA/skrypty/lab6> cp str2.c++ str3.c++
```

Wewnętrznie **string** w momencie przekroczenia rozmiaru prealokowanej tablicy, alokuje nową. Standardowo nowo alokowany jest rozmiar 2 razy większy niż aktualny. Spróbujmy z góry dokonać rezerwacji odpowiednio dużego rozmiaru stringa.

```
result.reserve(s.length());
```

Proszę skompilować procedurę z opcją, oraz bez opcji **-O2**.
Jaki jest czas działania?
Co pokazuje profiler?

2.4

Proszę wykonać kopię procedury

```
maciekw@Banach:~/studenci/OORA/skrypty/lab6> cp str3.c++ str4.c++
```

Przekazanie obiektu przez wartość jest nieefektywne. Oznacza wykonanie kopii, a co za tym idzie uruchomienie konstruktora na wejściu do funkcji. Przekażmy argument przez referencję.

```
std::string remove_ctrl(std::string const & s){
```

Proszę skompilować procedurę z opcją, oraz bez opcji **-O2**.
Jaki jest czas działania?
Co pokazuje profiler?

2.5

Proszę wykonać kopię procedury

```
maciekw@Banach:~/studenci/OORA/skrypty/lab6> cp str4.c++ str5.c++
```

W C++ iterator stringa jest wskaźnikiem. Może użycie iteratorów poprawi wydajność? W ten sposób 2 razy powinniśmy zmniejszyć ilość odwołań przez wskaźniki.

```
for (auto it = s.begin(), end = s.end(); it != end; it++){
        if ( *it >= 0x20 )
            result += *it;
```

Proszę skompilować procedurę z opcją, oraz bez opcji **-O2**.
Jaki jest czas działania?
Co pokazuje profiler?

2.6

Proszę wykonać kopię procedury

```
maciekw@Banach:~/studenci/OORA/skrypty/lab6> cp str5.c++ str6.c++
```

Zwracany **string** jest kopiowany! Zwróćmy wynik z stylu Fortan-owego **subroutine**. Proszę pamiętać o modyfikacji częśći **main**.

```
void remove_ctrl(std::string const & s, std::string & result){
    result.clear();
```

Proszę skompilować procedurę z opcją, oraz bez opcji **-O2**.
Jaki jest czas działania?
Co pokazuje profiler?

2.7

Proszę wykonać kopię procedury

```
maciekw@Banach:~/studenci/OORA/skrypty/lab6> cp str6.c++ str7.c++
```

Może rozwiązanie **bare metal** będie szybsze? Zrezygnujmy z zaawansowanych konstrukcji oferowanych przez C++ na rzecz prymitywnych tablic znaków z C.

```
void remove_ctrl(char const * s, char * result, size_t size){
    for (size_t i = 0; i < size; i++){
        if (s[i] >= 0x20)
            *result++ = s[i];
    }
    *result = 0;
}
...
char* results = (char*) malloc(sizeof(char) * s.length());
...
remove_ctrl(s.data(), results, s.length());
result = results;
....
  free(results);
```

Proszę skompilować procedurę z opcją, oraz bez opcji **-O2**.
Jaki jest czas działania?
Co pokazuje profiler?

2.8

Proszę wykonać kopię procedury

```
maciekw@Banach:~/studenci/OORA/skrypty/lab6> cp str1.c++ str8.c++
```

Użyjmy lepszego algorytmu!

```
std::string remove_ctrl(std::string s){
    std::string result;
    for (size_t begin = 0, i = begin, end = s.length();
    begin < end; begin = i + 1){
        for (i = begin; i < end; i++){
          if (s[i] < 0x20)
            break;
        }
        result = result + s.substr(begin, i - begin);
    }
```

```
    return result;
}
```

Proszę skompilować procedurę z opcją, oraz bez opcji **-O2**.
Jaki jest czas działania?
Co pokazuje profiler?

2.9

Proszę wykonać kopię procedury

```
maciekw@Banach:~/studenci/OORA/skrypty/lab6> cp str8.c++ str9.c++
```

Powtórzmy prealokację pamięci, oraz modyfikację wynikowego stringu zamiast ciągłego tworzenia nowych.

```
result.reserve(s.length());
...
result.append(s, begin, i - begin);
```

Proszę skompilować procedurę z opcją, oraz bez opcji **-O2**.
Jaki jest czas działania?
Co pokazuje profiler?

2.10

Proszę wykonać kopię procedury

```
maciekw@Banach:~/studenci/OORA/skrypty/lab6> cp str9.c++ str10.c++
```

Unikajmy kopiowania stringu wejściowego i wyjściowego.

```
std::string remove_ctrl(std::string const & s, std::string & result){
    result.clear();
```

Proszę skompilować procedurę z opcją, oraz bez opcji **-O2**.
Jaki jest czas działania?
Co pokazuje profiler?

2.11

Proszę wykonać kopię procedury

```
maciekw@Banach:~/studenci/OORA/skrypty/lab6> cp str10.c++ str11.c++
```

wykonajmy wersję bare metal powyższego.

```
#include <string.h>
...
void remove_ctrl(char const * s, char * result, size_t size){
    for (size_t begin = 0, i = begin, end = size;
    begin < end; begin = i + 1){
        for (i = begin; i < end; i++){
          if (s[i] < 0x20)
            break;
        }
        memcpy (result, s + begin, i - begin);
        result += i - begin;
    }
    *result = 0;
}
...
```

Proszę skompilować procedurę z opcją, oraz bez opcji **-O2**.
Jaki jest czas działania?
Co pokazuje profiler?

3 Podsumowanie

Które metody optymalizacji dały najlepsze rezultaty?
Dlaczego? Jakie mechanizmy za nimi stały?

Literatura

[1] Kurt Gunteroth, *Optimized C++*, O'Reilly, 2016