Кафедра прикладной математики и кибернетики

Выполнили:

Студенты 3 курса группы ИП-111  
Корнилов А.А.,  
Попов М.И.,

Толкач А.А.

Проверил:

Профессор кафедры ПМиК  
Малков Е.А.

ЛАБОРАТОРНАЯ РАБОТА №8

По дисциплине: «Операционные системы»

Сибирский государственный университет  
телекоммуникаций и информатики

Новосибирск, 2023

Министерство цифрового развития, связи  
и массовых коммуникаций Российской Федерации

**Задание:** программно реализуйте вычисление суммы последовательности чисел на основе последовательного кода, интерфейсов Pthreads и C++11 <thread>. Сравните время вычислений.

**Цель:** знакомство с программными интерфейсами управления потоками в Linux

**Выполнение работы:**

В качестве программы для вычисления суммы последовательности чисел была написана программа для вычисления числа Pi при помощи ряда Лейбца с точностью до 64 знаков после запятой (long double) на языках C (лист. 1) и C++ (лист. 2).

|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  #include <math.h>  #include <time.h>  int main() {  clock\_t start = clock();  long double pi = 0;  int j = 1;  const int prec = 64;  for (int i = 1; i < 500000000; i++) {  pi += (double)4 / j;  j += 2;  pi -= (double)4 / j;  j += 2;  }  printf("w = %2d p = %2d pi = %\*.\*Lf\n", 62, 62, 62, 62, pi);  clock\_t end = clock();  printf("%s%f%s\n","Потрачено на вычисление и вывод: ",(double)(end - start) / CLOCKS\_PER\_SEC, " сек.");  return 0;  } |

Листинг 1 – программа lab08\_5.c

|  |
| --- |
| #include <iostream>  #include <iomanip>  #include <time.h>  using namespace std;  int main(){  clock\_t start = clock();  long double pi = 0;  int j = 1;  const int prec = 64;  for (int i = 1; i < 500000000; i++){  pi += (double)4 / j;  j += 2;  pi -= (double)4 / j;  j += 2;  }  cout << setprecision(prec) << pi << endl;  clock\_t end = clock();  cout << "Потрачено на вычисление и вывод: " << setprecision(3) << (double)(end - start) / CLOCKS\_PER\_SEC << " сек." << endl;  return 0;  } |

Листинг 2 – программа lab08\_6.cpp

Компиляция программы и вывод для С:

|  |
| --- |
| miron@DESKTOP-UMC1Q46:/mnt/u/Documents/В ВУЗ/OS/8$ gcc lab08\_5.c -o lab08\_5  miron@DESKTOP-UMC1Q46:/mnt/u/Documents/В ВУЗ/OS/8$ ./lab08\_5  w=62 p=62  pi = 3.14159265258979539052995588743755206451169215142726898193359375  Потрачено на вычисление и вывод: 6.618006 сек. |

Компиляция программы и вывод для С++:

|  |
| --- |
| miron@DESKTOP-UMC1Q46:/mnt/u/Documents/В ВУЗ/OS/8$ g++ lab08\_6.cpp -o lab08\_6  miron@DESKTOP-UMC1Q46:/mnt/u/Documents/В ВУЗ/OS/8$ ./lab08\_6  3.14159265258979539052995588743755206451169215142726898193359375  Потрачено на вычисление и вывод: 6.63 сек. |

Позже в отчете будет приведена таблица (табл.1) для подсчетов точности, времени и правильности относительно реального числа pi.

Теперь модифицируем алгоритм программы lab08\_5.c для параллельного вычисления при помощи библиотеки pthread.h (лист. 3). Вычисления будут проводиться на 4 потоках заданных NUM\_THREADS и количество итераций ITERATIONS. Сама функция вычисления calculate\_pi устроена так что делит количество вычисляемый итераций относительно числу потоков и при помощи mutex-ов осуществляться разделение доступа к числу pi от локального вычисляемого потоком local\_pi.

|  |
| --- |
| #include <stdio.h>  #include <pthread.h>  #include <time.h>  #define NUM\_THREADS 4  #define ITERATIONS 500000000  long double pi = 0;  pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;  void\* calculate\_pi(void\* thread\_id) {  long double local\_pi = 0;  int thread\_num = \*(int\*)thread\_id;  int start = thread\_num \* (ITERATIONS / NUM\_THREADS) + 1;  int end = (thread\_num + 1) \* (ITERATIONS / NUM\_THREADS);  for (int I = start; I <= end; i++) local\_pi += (I % 2 == 1) ? (long double)4 / (2 \* I – 1) : -(long double)4 / (2 \* I – 1);  pthread\_mutex\_lock(&mutex);  pi += local\_pi;  pthread\_mutex\_unlock(&mutex);  pthread\_exit(NULL);  }  int main() {  clock\_t start = clock();  pthread\_t threads[NUM\_THREADS];  int thread\_ids[NUM\_THREADS];  for (int I = 0; I < NUM\_THREADS; i++) {  thread\_ids[i] = I;  pthread\_create(&threads[i], NULL, calculate\_pi, (void\*)&thread\_ids[i]);  }  for (int I = 0; I < NUM\_THREADS; i++) pthread\_join(threads[i], NULL);  printf(“w = %2d p = %2d pi = %\*.\*Lf\n”, 62, 62, 62, 62, pi);  clock\_t end = clock();  printf(“%s%f%s\n”, “Потрачено на вычисление и вывод: “, (double)(end – start) / CLOCKS\_PER\_SEC, “ сек.”);  pthread\_mutex\_destroy(&mutex);  return 0;  } |

Листинг 3 – программа lab08\_5p.cpp

Компиляция программы и вывод:

|  |
| --- |
| miron@DESKTOP-UMC1Q46:/mnt/u/Documents/В ВУЗ/OS/8$ gcc lab08\_5p.c -lpthread -o lab08\_5p  miron@DESKTOP-UMC1Q46:/mnt/u/Documents/В ВУЗ/OS/8$ ./lab08\_5p  w = 62 p = 62 pi = 3.14159265158979430143312838730196290271123871207237243652343750  Потрачено на вычисление и вывод: 3.410105 сек. |

Время вычисления сократилось вдвое.

Также модифицируем программу lab08\_6.c написанную на языке С++, для использования параллельности будем использовать библиотеку thread (лист. 4). Вместо стандартной переменной long double будем использовать атомарную переменную типа double (методы для long double по неизвестной причине не доступны), барьерное разделение вычислений теперь будет проходить в main функции программы по такому-же принципу.

|  |
| --- |
| #include <iostream>  #include <iomanip>  #include <thread>  #include <vector>  #include <atomic>  #include <ctime>  using namespace std;  const int num\_threads = 4; // Количество потоков для параллельных вычислений  const int iterations\_per\_thread = 500000000; // Количество итераций на каждом потоке  atomic<double> pi{ 0.0 }; // Используем атомарную переменную для суммирования  void calculatePi(int start, int end) {  double localSum = 0;  int j = 2 \* start + 1;  for (int I = start; I < end; i++){  if (I % 2 == 0) {  localSum += 4.0 / j;  }  else {  localSum -= 4.0 / j;  }  j += 2;  }  pi.store(localSum, memory\_order\_relaxed);  }  int main() {  clock\_t start\_time = clock();  vector<thread> threads;  for (int I = 0; I < num\_threads; i++) {  int start = I \* iterations\_per\_thread;  int end = start + iterations\_per\_thread;  threads.push\_back(thread(calculatePi, start, end));  }  for (auto& t : threads) t.join();  cout << setprecision(64) << pi << endl;  clock\_t end\_time = clock();  cout << “Потрачено на вычисление и вывод: “ << setprecision(3) << (double)(end\_time – start\_time) / CLOCKS\_PER\_SEC << “ сек.” << endl;  return 0;  } |

Компиляция программы и вывод:

|  |
| --- |
| miron@DESKTOP-UMC1Q46:/mnt/u/Documents/В ВУЗ/OS/8$ g++ lab08\_6p.cpp -o lab08\_6p  miron@DESKTOP-UMC1Q46:/mnt/u/Documents/В ВУЗ/OS/8$ ./lab08\_6p  3.141592645589323939958603659761138260364532470703125  Потрачено на вычисление и вывод: 1.64 сек. |

Время работы сократилось в 4-е раза, но программа не всегда выдает точный результат, этот был получен с 6-ой попытки.

В таблице (табл. 1) приведены все подсчеты программы с учетом использование параллельности и использованием разных компиляторов. Точность указывается от количества полученных в ответе чисел, в расхождении указывается после какого числа начинаться отличия по сравнением с оригинальным числом.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Компилятор | Файл | Время выполнения, сек. | Результат | Точность | Расхождение |
| Оригинал числа с Wiki | | | 3.14159265358979323846264338327950288419716939937510582097494459 | 63 | 63 |
| GCC | lab08\_5.c | 6.62 | 3.14159265258979539052995588743755206451169215142726898193359375 | 64 | 15 |
| G++ | lab08\_6.c | 6.63 | 3.14159265258979539052995588743755206451169215142726898193359375 | 63 | 15 |
| MVS CL | lab08\_6.c | 3.24 | 3.1415926525880504271981408237479627132415771484375 | 50 | 9 |
| GCC | lab08\_5p.c | 3.41 | 3.14159265158979430143312838730196290271123871207237243652343750 | 63 | 9 |
| G++ | lab08\_6p.c | 1.64 | 3.141592645589323939958603659761138260364532470703125 | 52 | 8 |
| MVS CL | lab08\_6p.c | 0.467 | 3.141592645589323939958603659761138260364532470703125 | 52 | 9 |

Таблица 1 – Результаты работ программы и их подсчеты

Вывод: С разделением программ на для параллельного вычисления точность вычислений сильно падает, быстрейшими компилятором для программ оказался MVS CL. Вычисления проводились с Windows 11 и оболочной WSL.