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 №10

По дисциплине: «Операционные системы»

Сибирский государственный университет  
телекоммуникаций и информатики

Новосибирск, 2023

Министерство цифрового развития, связи  
и массовых коммуникаций Российской Федерации

**Задание:** протестируйте программы лабораторных 8 и 9, используя программную реализацию алгоритма Петерсона, запуская их на одном, двух и нескольких ядрах. Протестируйте модифицированный на основе атомарных функций код алгоритма Петерсона используя различные модели упорядочения выполнения инструкций кода

**Цель:** знакомство с атомарными функциями.

**Выполнение работы:**

В качестве программы использован модифицированная версия программы 8 лабораторной работы. Алгоритм Петерсона реализован через использование атомарных переменных flag, turn и sum.

|  |
| --- |
| #include <iostream>  #include <iomanip>  #include <thread>  #include <atomic>  using namespace std;  const long long num\_steps = 1000000000; // количество шагов для вычисления  const double step = 1.0 / static\_cast<double>(num\_steps);  std::atomic<double> sum{ 0.0 };  std::atomic<int> flag{ 0 };  std::atomic<int> turn{ 0 };  void calculatePi(int id, int num\_threads) {  double x;  double partial\_sum = 0.0;  for (int i = id; i < num\_steps; i += num\_threads) {  x = (i + 0.5) \* step;  partial\_sum += 4.0 / (1.0 + x \* x);  }  partial\_sum \*= step;  // Вход в критическую секцию  flag.store(id, memory\_order\_relaxed);  turn.store(num\_threads - 1 - id, memory\_order\_relaxed);  for (int j = 0; j < num\_threads; ++j) {  while ((j != id) and (flag.load(memory\_order\_relaxed) == id) and (turn.load(memory\_order\_relaxed) == num\_threads - 1 - id)) {  // Ждем своей очереди  this\_thread::yield();  }  }  sum.fetch\_add(partial\_sum, memory\_order\_relaxed);  // Выход из критической секции  flag.store(-1, memory\_order\_relaxed);  }  int main() {  int num\_threads = thread::hardware\_concurrency();  cout << "num\_threads = " << num\_threads << endl;  //int num\_threads = 1;  thread threads[num\_threads];  // Создание потоков для вычислений  for (int i = 0; i < num\_threads; ++i) threads[i] = thread(calculatePi, i, num\_threads);  // Ожидание завершения потоков  for (int i = 0; i < num\_threads; ++i) threads[i].join();  cout << "Реальное число pi: 3.14159265358979323846264338327950288419716939937510582097494459" << endl;  cout << setprecision(64) << "Вычисляемое число pi: " << sum.load(memory\_order\_relaxed) << endl;  return 0;  } |

Листинг 1 – программа lab10\_3.c

При входе в критическую секцию каждый поток перед входом в критическую секцию устанавливает свой индекс в переменной flag и обозначает свою очередь в переменной turn. Эти операции (flag.store() и turn.store()) являются атомарными, чтобы избежать состязания за общие ресурсы. После установки своей очереди в переменной turn, поток ожидает, пока не наступит его очередь для входа в критическую секцию. Ожидание происходит в цикле while, проверяя условия, что это именно тот поток, который установил свою очередь, и что переменная flag указывает на него. После выполнения критической секции (в данном случае, обновления sum), поток устанавливает переменную flag в -1, обозначая, что он покинул критическую секцию. Это позволяет другим потокам войти в критическую секцию, если их очередь наступила. Переменные flag и turn используются для управления тем, какой поток может войти в критическую секцию. Последовательность их обновлений обеспечивает взаимное исключение. После выхода из критической секции поток обновляет переменную flag в -1, чтобы показать, что он покинул критическую секцию, и ожидает своей очереди, прежде чем вернуться к выполнению.