МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Отчёт по лабораторной работе №4 по дисциплине «Параллельное программирование»

**Синхронизация потоков**

|  |  |
| --- | --- |
| Выполнил: студент группы ФИб-4302-51-00 | / Д.А. Савин / |
| Проверил: ст. преподаватель каф. ПМиИ | / В.А. Бызов / |

Киров 2021

Задание 1

«Производители – потребители». Два потока обрабатывают заявки. Первый поток читает заявку и помещает её в очередь размера N, если очередь переполнена, то заявке отказывается. Второй поток берет заявки из очереди и обрабатывает их, если в очереди заявок нет, то поток засыпает. Написать параллельную программу, реализующую правильную работу этих потоков.

Код:

|  |
| --- |
| #include <iostream>  #include <queue>  #include <thread>  #include <mutex>  using namespace std;  #define queueMaxSize 10  mutex mx;  queue <int> requests;  void SecondThread()  {  while (true)  {  if (!requests.empty())  {  mx.lock();  int request = requests.front();  requests.pop();  mx.unlock();  this\_thread::sleep\_for(chrono::milliseconds(request));  cout << "запрос " << request << " обработан" << endl;  }  }  }  int main()  {  setlocale(0, "");  thread thread\_ = thread(SecondThread);  thread\_.detach();  int n = 1;  while (n > 0)  {  cout << ("введите заявку: ");  try  {  cin >> n;  }  catch (const std::exception&)  {  continue;  }  if (requests.size() >= queueMaxSize)  {  cout << ("очередь переполнена. заявка не добавлена.") << endl;  continue;  }  mx.lock();  requests.push(n);  cout << "заявка " << n << " добавлена." << endl;  mx.unlock();  }  cin.get();  } |

Задание 2

«Фоновая обработка». Два потока обрабатывают файл, содержащий набор целочисленных массивов. В файле находится M массивов, каждый в отдельной строке. В первой строке указано число M – количество массивов, а в следующих M строках записаны элементы массивов, разделенные пробелами. Первый поток по очереди читает массивы из файла, а второй по очереди считает суммы элементов этих массивов. Результаты первый поток должен записать в новый файл. Каждую сумму – в отдельной строке. Организовать правильную работу потоков.

Код:

|  |
| --- |
| #include "pch.h"  #include <iostream>  #include <fstream>  #include <string>  #include <vector>  #include <thread>  #include <mutex>  using namespace std;  int countOfArrays = 0;  int sumByLine = 0;  string inputFileName = "input.txt";  string outputFileName = "output.txt";  vector <string> numbers;  vector <int> sum;  mutex inputMutex, outputMutex;  thread thread1, thread2;  string line = "";  void calcSum();  void writingToFile(string outputFileName);  void readingFromFile(string inputFileName)  {  cout << "Первый поток стартовал. " << endl;  ifstream in;  int numberOfLinesRead = 0;  in.open(inputFileName);  getline(in, line);  countOfArrays = stoi(line);  inputMutex.lock();  while (numberOfLinesRead != countOfArrays)  {  getline(in, line);  numbers.push\_back(line);  numberOfLinesRead++;  }  inputMutex.unlock();  while (sum.size() == 0) {}  outputMutex.lock();  writingToFile(outputFileName);  outputMutex.unlock();  cout << "Первый поток закончился. " << endl;  in.close();  }  void writingToFile(string outputFileName)  {  ofstream myfile;  myfile.open(outputFileName, std::ios::app);  for (int i = 0; i < sum.size(); i++)  {  myfile << sum[i] << "\n";  }  myfile.close();  }  //sum += atoi(string({ (char)strV[i][j] }).c\_str());  void calcSum()  {  while (numbers.size()==0) { }  cout << "Второй поток стартовал. " << endl;  char\* ptr = 0;  char\* arr = new char[numbers[0].size()];  outputMutex.lock();  for (int i = 0; i < numbers.size(); i++)  {  sumByLine = 0;  strcpy(arr, numbers[i].c\_str());  //inputMutex.lock();  ptr = strtok(arr, " ");  //inputMutex.unlock();  while (ptr)  {  sumByLine += atoi(ptr);  ptr = strtok(0, " ");  }  sum.push\_back(sumByLine);  }  cout << "Второй поток закончился." << endl;  outputMutex.unlock();  }  int main()  {  setlocale(LC\_ALL, "rus");  thread2 = thread(calcSum);  readingFromFile(inputFileName);  //thread1 = thread(readingFromFile, inputFileName);  //thread1.join();  thread2.detach();    //cin.get();  } |

Задание 3

«Читатели – писатели». Имеется хранилище данных, с которым работают одновременно несколько потоков. Первые N потоков случайным образом изменяют данные в хранилище (при этом изменять данные в некоторый момент времени может ровно один поток), остальные M потоков периодически читают данные из хранилища. Несколько читателей могут находиться в хранилище одновременно, но читатель и писатель не могут одновременно использовать хранилище. Эмулировать работу хранилища.

Код:

|  |
| --- |
| #include <iostream>  #include <queue>  #include <iostream>  #include <thread>  #include <mutex>  #include <random>  using namespace std;  #define READERS\_COUNT 2  #define WRITTERS\_COUNT 2  std::mutex readerCountAcces;  int storage = 0;  int readerCount = 0;  //bool readSem = true;  //bool access = true;  #define queueMaxSize 10  std::mutex mx;  mutex readSem;  mutex access;  void ReaderThread(int threadId)  {  while (true)  {  //mx.lock();  if (readerCount != -1)  {  readerCount++;  cout << "reader thread #" << threadId << " read this value: " << storage << " current readers: " << readerCount << endl;  //mx.unlock();  std::this\_thread::sleep\_for(std::chrono::milliseconds(rand() % 20000));  mx.lock();  readerCount--;  mx.unlock();  }  //else mx.unlock();  std::this\_thread::sleep\_for(std::chrono::milliseconds(1000 + rand() % 10000));  }  }  void WriterThread(int threadId)  {  while (true)  {  mx.lock();  if (readerCount == 0)  {  readerCount = -1;  mx.unlock();  int oldValue = storage;  storage = rand() % 100;  cout << "writer thread #" << threadId << " changed value from " << oldValue << " to " << storage << endl;  mx.lock();  readerCount = 0;  mx.unlock();  }  else mx.unlock();  std::this\_thread::sleep\_for(std::chrono::milliseconds(1000 + rand() % 10000));  }  }  int main()  {  srand(time(NULL));  setlocale(0, "");  vector<thread> readersThreads;  int idCounter = 0;  for (int i = 0; i < READERS\_COUNT; i++)  {  readersThreads.push\_back(thread(ReaderThread, i));  }  vector<thread> writersThreads;  for (int i = 0; i < WRITTERS\_COUNT; i++)  {  writersThreads.push\_back(thread(WriterThread, i));  }  for (int i = 0; i < READERS\_COUNT; i++)  {  readersThreads[i].join();  }  for (int i = 0; i < WRITTERS\_COUNT; i++)  {  writersThreads[i].join();  }  cin.get();  } |

Вывод

Для синхронизации могут использоваться различные способы, методы и классы: mutex, lock. Правильная синхронизация позволяет сократить время простоя и ожидания, а также максимально эффективно использовать вычислительные ресурсы системы.