Федеральное государственное бюджетное образовательное учреждение высшего образования «**Московский государственный** **технический** **университет** **имени**

Н. Э. **Баумана**

Факультет информатики и систем управления

Кафедра ИУ9 «Теоретическая информатика и компьютерные технологии»

Факультет: Информатика и системы управления

Кафедра: Теоретическая информатика и компьютерные технологии

Лабораторная работа №4  
«Задача о пяти обедающих философах»

по курсу: «Разработка параллельных и распределенных программ»

Выполнила: Студентка группы ИУ9-51Б

Александрова О.С

Проверил: Царев А. С.

Задание:

Изображение выглядит как текст, Шрифт, снимок экрана, черно-белый

Автоматически созданное описание

Характеристики устройства, на котором производились вычисления

Название модели: MacBook Air

Идентификатор модели: MacBookAir10,1

Чип: Apple M1

Общее количество ядер: 8 (4 производительности и 4 эффективности)

Память: 8 ГБ

Ход работы (листинг текста программы)

#include <iostream>  
#include <thread>  
#include <chrono>  
#include <fstream>  
#include <atomic>  
#include <string>  
#include <vector>  
#include <sstream>  
using namespace std;  
  
void states(int n);  
ofstream file("philosopher.txt");  
mutex printMutex;  
atomic<bool> running{true};  
  
class Philosopher {  
public:  
 string name;  
 mutex& Leftfork;  
 mutex& Rightfork;  
  
 Philosopher(string xname, mutex& Leftfork, mutex& Rightfork) :  
 name(xname), Leftfork(Leftfork), Rightfork(Rightfork) {}  
  
 void run() {  
 while (running) {  
 this\_thread::sleep\_for(chrono::milliseconds(rand() % 10 + 1));  
 takeForks();  
 }  
 }  
  
 void takeForks() {  
  
 if (Leftfork.try\_lock()) {  
 printToFile(name + " БеретЛевуюВилку");  
 if (Rightfork.try\_lock()) {  
 lock\_guard<mutex> leftLock(Leftfork, adopt\_lock);  
 printToFile(name + " БеретПравуюВилку");  
 lock\_guard<mutex> rightLock(Rightfork, adopt\_lock);  
 eating();  
 printToFile(name + " ПоложилЛевуюВилку");  
 printToFile(name + " ПоложилПравуюВилку");  
// Rightfork.unlock();  
 } else {  
 Leftfork.unlock();  
 }  
 }  
 }  
  
  
 void printToFile(const string& message) {  
 lock\_guard<mutex> lock(printMutex);  
 file << message << endl;  
 }  
  
 void eating() {  
 {  
 lock\_guard<mutex> lock(printMutex);  
 file << name << " НачинаетEсть" << endl;  
 }  
 this\_thread::sleep\_for(chrono::milliseconds(rand() % 10 + 1));  
 {  
 lock\_guard<mutex> lock(printMutex);  
 file << name << " ЗаканчиваетЕсть" << endl;  
 }  
 }  
};  
  
void Philosophers(int numOfPhilosophers) {  
 mutex forks[numOfPhilosophers];  
 string philosopherNames[] = { "1", "2", "3", "4", "5" };  
 vector<Philosopher> philosophersVec;  
  
 for (int i = 0; i < numOfPhilosophers; i++) {  
 philosophersVec.emplace\_back(Philosopher(philosopherNames[i], forks[i], forks[(i + 1) % numOfPhilosophers]));  
 }  
  
 vector<thread> threads;  
 for (int i = 0; i < numOfPhilosophers; i++) {  
 threads.emplace\_back(&Philosopher::run, &philosophersVec[i]);  
 }  
  
 this\_thread::sleep\_for(chrono::seconds(5));  
 running = false;  
  
 for (int i = 0; i < numOfPhilosophers; i++) {  
 threads[i].join();  
 }  
}  
  
int main() {  
 int numOfPhilosophers;  
 cout << "Enter the number of philosophers: ";  
 cin >> numOfPhilosophers;  
 Philosophers(numOfPhilosophers);  
  
 states(numOfPhilosophers);  
  
 return 0;  
}  
  
void states(int n) {  
 std::ifstream prv\_file("philosopher.txt");  
 std::ofstream res\_file("states.txt");  
// int n = 5;  
 int num\_strngs = 0;  
 std::string line;  
 while (std::getline(prv\_file, line)) {  
 num\_strngs += 1;  
 }  
 prv\_file.clear(); // Переходим в начало файла снова  
 prv\_file.seekg(0, std::ios::*beg*);  
  
 int k = 0;  
 std::vector<std::vector<std::string>> status(num\_strngs, std::vector<std::string>(n, "Размышляет"));  
 while (std::getline(prv\_file, line)) {  
 std::istringstream iss(line);  
 int ind;  
 std::string state;  
 iss >> ind >> state;  
 status[k][ind-1] = state;  
 if (k > 0) {  
 for (int j = 0; j < n; j++) {  
 if (j != ind-1) {  
 if (status[k-1][j] == "ПоложилПравуюВилку") {  
 status[k][j] = "Размышляет";  
 } else if (status[k-1][j] == "НачинаетЕсть") {  
 status[k][j] = "ест";  
 } else {  
 status[k][j] = status[k-1][j];  
 }  
 }  
 }  
 }  
 k += 1;  
 }  
  
 for (int i = 0; i < num\_strngs; i++) {  
 res\_file << i << " момент ";  
 for (int j = 0; j < n; j++) {  
 res\_file << status[i][j] << "(" << j+1 << ") ";  
 }  
 res\_file << "\n";  
 }  
}

Количество философов вводится с клавиатуры и является не привязанным к какому-либо N. Для тестирования программы было выбрано N = 5:

int main() {  
 int numOfPhilosophers;  
 cout << "Enter the number of philosophers: ";  
 cin >> numOfPhilosophers;  
 Philosophers(numOfPhilosophers);  
  
 states(numOfPhilosophers);  
  
 return 0;  
}

Результат работы алгоритма

Текущие состояния каждого философа выводятся в файл philosophers.txt

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Затем данные анализируются отдельной функции и состояние каждого философа в каждый момент времени можно увидеть в другом файле:

Изображение выглядит как снимок экрана, текст, шаблон

Автоматически созданное описание

Вывод

Были изучены и реализованы способы работы с mutex и накладывание блокировки на конкретной задаче. Реализовано исключения состояния взаимоблокировки : в методе

takeForks() философы пытаются взять вилки сначала поочередно, а затем, если не могут взять правую вилку, отпускают левую, чтобы избежать ситуации, когда один философ держит левую вилку и ждёт правую, которую удерживает другой философ.