**

Государственное образовательное учреждение высшего профессионального образования «Московский Государственный Технический Университет

имени Н.Э. Баумана»

**ОТЧЕТ**

По лабораторной работе №5

По курсу «Анализ алгоритмов»

Тема: «Алгоритм конвейерной обработки»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | Студент: | | | Жарова Е.А. | |
|  | | Группа | | | ИУ7-51 | |
|  | |  | | |  | |
|  | |  | |  | | |
|  | | Москва, 2017 | | |  | |

Оглавление

[Постановка задачи 1](#_Toc509193769)

[Описание алгоритма 1](#_Toc509193770)

[Реализация 1](#_Toc509193771)

[Кольцевой буфер 1](#_Toc509193772)

[Этап генерации чисел 2](#_Toc509193773)

[Этап суммирования 2](#_Toc509193774)

[Этап вывода 3](#_Toc509193775)

[Основная программа 3](#_Toc509193776)

[Заключение 3](#_Toc509193777)

# **Постановка задачи**

Необходимо изучить и реализовать алгоритм конвейерной обработки данных.

# **Описание алгоритма**

Алгоритмом симулируется работа конвейера, состоящая из трёх этапов:

1. Генерация случайных чисел
2. Сложение нескольких чисел
3. Печать вычисленной суммы

Между этими тремя этапами осуществляется обмен при помощи кольцевого буфера.

Работа конвейера симулируется при помощи трёх потоков, которые заполняют или добавляют числа в необходимый буфер.

# **Реализация**

## Кольцевой буфер

template <typename T, size\_t Size>

class RingBuffer

{

private:

T buff[Size];

size\_t front;

size\_t back;

size\_t count;

public:

RingBuffer()

: front(0),

back(0),

count(0)

{}

bool push(T data)

{

if(isFull())

return false;

buff[front] = data;

front++;

count++;

front %= Size;

return true;

}

bool pop(T& x)

{

if(isEmpty())

return false;

x = buff[back];

back++;

count--;

back %= Size;

return true;

}

size\_t getCount() const

{

return count;

}

bool isEmpty() const

{

return count == 0;

}

bool isFull() const

{

return count == Size;

}

void print() const

{

//std::cout << "Empty:" << isEmpty() << std::endl;

//std::cout << "Full:" << isFull() << std::endl;

//std::cout << "Count:" << getCount()<< std::endl;

size\_t i = back;

if (count != 0)

{

do {

std::cout << buff[i] << " ";

i++;

i %= Size;

} while(i != front);

std::cout << std::endl;

}

std::cout << std::endl;

}

};

## Этап генерации чисел

template <typename T>

void generateNewData(RingBuffer<T, GEN\_BUFSIZE> &buff, timeType sleep)

{

while(1) {

std::this\_thread::sleep\_for(std::chrono::milliseconds(sleep));

if(!buff.isFull()) {

buff.push(rand() % 3);

}

}

}

## Этап суммирования

template <typename T>

void sumData(RingBuffer<T, SUM\_BUFSIZE> &sumBuff, RingBuffer<T, GEN\_BUFSIZE> &buff, timeType sleep, size\_t sumCount)

{

while(1) {

std::this\_thread::sleep\_for(std::chrono::milliseconds(sleep));

if(buff.getCount() >= sumCount) {

T sum = 0;

for(size\_t i = 0; i < sumCount; i++) {

T x;

buff.pop(x);

sum += x;

}

while(sumBuff.isFull()) {}

sumBuff.push(sum);

}

}

}}

## Этап вывода

template <typename T>

void printData(RingBuffer<T, SUM\_BUFSIZE> &sumBuff, timeType sleep)

{

while(1) {

std::this\_thread::sleep\_for(std::chrono::milliseconds(sleep));

//cout << "1 ";

if(!sumBuff.isEmpty()) {

T x;

sumBuff.pop(x);

//system("sl");

cout << x << " " << std::flush;// << endl;

//std::flush = true;

}

}

}

## Основная программа

int \_tmain(int argc, \_TCHAR\* argv[])

{

RingBuffer<int, GEN\_BUFSIZE> generBuff;

RingBuffer<int, SUM\_BUFSIZE> sumBuff;

std::thread generator(&generateNewData<int>, std::ref(generBuff), 10);

std::thread summator(&sumData<int>, std::ref(sumBuff), std::ref(generBuff), 20, 2);

std::thread printator(&printData<int>, std::ref(sumBuff), 1);

generator.join();

summator.join();

printator.join();

return 0;

}

# **Заключение**

Был изучен и реализован алгоритм конвейерной обработки данных.