Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

Московский государственный технический университет имени Н.Э.Баумана
(МГТУ им. Н.Э.Баумана)

ОТЧЕТ
По лабораторной работе № 5
По курсу «Анализ алгоритмов»
на тему «Алгоритм конвейерной обработки»

Выполнил Студент: Московец Н.С
Группа: ИУ7-51

Москва, 2017

Оглавление

Постановка задачи

Изучить и реализовать конвейерный алгоритм обработки данных.

Описание алгоритма

Алгоритм симулирует работу конвейера, который состоит из 3-х этапов. На первом этапе алгоритм генерирует случайные числа, на втором складывает несколько чисел, на третьем - печатает вычисленную сумму.

Между этапами осуществляется обмен данных с помощью кольцевых буферов.

Работа конвейера симулируется с помощью трех 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
    {
        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)
3. {
4.     while(1) {
5.         std::this_thread::sleep_for(std::chrono::milliseconds(sleep));
6.
7.         if(!buff.isFull()) {
8.             buff.push(rand() % 3);
9.         }
10.        }
11.    }
```

Этап суммирования

```
1. template <typename T>
2. void sumData(RingBuffer<T, SUM_BUFSIZE> &sumBuff,
3. RingBuffer<T, GEN_BUFSIZE> &buff, timeType sleep, size_t sumCount)
4. {
5.     while(1) {
6.         std::this_thread::sleep_for(std::chrono::milliseconds(sleep));
7.
8.         if(buff.getCount() >= sumCount) {
9.             T sum = 0;
10.                for(size_t i = 0; i < sumCount; i++) {
11.                    T x;
12.                    buff.pop(x);
13.                    sum += x;
14.                }
15.                while(sumBuff.isFull()) {}
16.                sumBuff.push(sum);
17.            }
18.        }
19.    }
```

Этап вывода

```
1.
2. template <typename T>
3. void printData(RingBuffer<T, SUM_BUFSIZE> &sumBuff, timeType sleep)
4. {
5.     while(1) {
6.         std::this_thread::sleep_for(std::chrono::milliseconds(sleep));
7.         if(!sumBuff.isEmpty()) {
8.             T x;
9.             sumBuff.pop(x);
10.                cout << x << " " << std::flush;
11.            }
12.        }
13.    }
```

Основная программа:

```
1. int main(int argc, char *argv[])
2. {
3.     RingBuffer<int, GEN_BUFSIZE> generBuff;
4.     RingBuffer<int, SUM_BUFSIZE> sumBuff;
5.
6.     std::thread generator(&generateNewData<int>, std::ref(generBuff),
   10);
7.     std::thread summator(&sumData<int>, std::ref(sumBuff),
   std::ref(generBuff), 20, 2);
8.     std::thread printator(&printData<int>, std::ref(sumBuff), 1);
9.
10.        generator.join();
11.        summator.join();
12.        printator.join();
13.         return 0;
14.    }
```

Заключение

Во время выполнения работы был изучен и реализован алгоритм конвейерной обработки данных.