

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное учреждение высшего образования
«Чувашский государственный университет И.Н. Ульянова»
Факультет информатики и вычислительной техники
Кафедра вычислительной техники

Параллельное программирование
Лабораторная работа 4
Выполнение заданий с 21 - 26

Выполнил:

Студент группы ИВТ-41-20
Галкин Д.С.

Проверил:

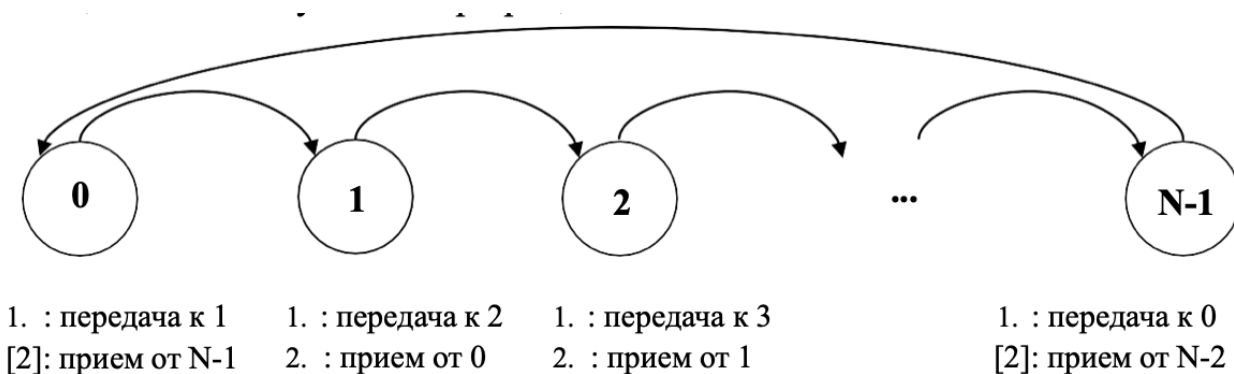
Ковалев С.В.

Цель работы (Технология программирования в MPI):

Задание 21. Коммуникация "точка-точка" (схема "сдвиг" по кольцу):

Задание для выполнения

Напишите MPI-программу, реализующую при помощи блокирующих функций отправки сообщений типа точка-точка схему коммуникации процессов "сдвиг по кольцу", в которой осуществляются одновременные отправка и прием сообщений всеми процессами. В качестве передаваемого сообщения используйте номер процесса.



- Процесс с номером X

[t] - Последовательность выполнения
действий, где t – порядковый номер

N - Общее количество процессов

Входные данные: нет

Выходные данные: "[номер_процесса]: receive message '<сообщение>'"

| Входные данные | Выходные данные (4 процесса) |
|----------------|------------------------------|
|----------------|------------------------------|

| | |
|--|--------------------------|
| | 1. : receive message '3' |
| | 2. : receive message '0' |
| | 3. : receive message '1' |
| | 4. : receive message '2' |

Полный текст программы:

1. Класс Main

```
var lab7 = new Lab7("Lab7", args);  
lab7.Start();
```

2. Класс Lab7

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using MPI;
using ProgramMPI.Interfaces;

namespace ProgramMPI.Labs
{
    public class Lab7 : ILab
    {
        public string Name { get; set; }
        public string[] _args { get; set; }

        public Lab7(string name, string[] args)
        {
            Name = name;
            _args = args;
        }

        public void ProccessStart(string[] args)
        {
            using (new MPI.Environment(ref args))
            {
                Intracommunicator comm = Communicator.world;

                int rank = comm.Rank;
                int size = comm.Size;

                // Определение номера процесса, которому нужно отправить сообщение
                int dest = (rank + 1) % size;
                // Определение номера процесса, от которого нужно принять сообщение
                int source = (rank - 1 + size) % size;

                // Отправка номера своего процесса следующему процессу в кольце

                comm.Send(rank, dest, 0);

                // Получение сообщения от предыдущего процесса в кольце
                int receivedRank = comm.Receive<int>(source, 0);

                Print(comm.Rank, receivedRank);
            }
        }

        public void Start()
        {
            Console.WriteLine(this);

            ProccessStart(_args);

            Console.WriteLine("\n");
        }
    }
}
```

```
public override string ToString() => $"{Name} started classes:";

private void Print(int rank, int receivedMessage) =>
    Console.WriteLine($"{rank + 1}: received message: '{receivedMessage}'");

    }
}
```

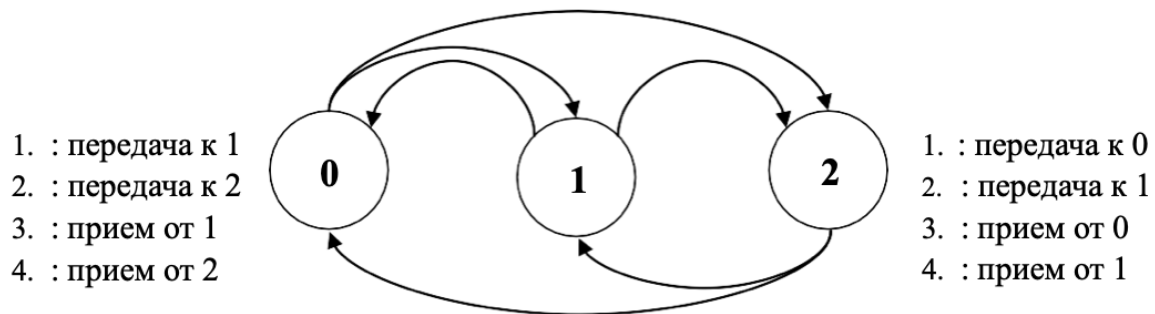
Результат:

```
Lab7 started classes:
[2]: received message: '0'
[3]: received message: '1'
[1]: received message: '3'
[4]: received message: '2'
```


Задание 22. Коммуникации "точка-точка" (каждый каждому):

Задание для выполнения

Напишите MPI-программу, реализующую при помощи блокирующих функций отправки сообщений типа точка-точка схему коммуникации процессов «каждый каждому», в которой осуществляется пересылка сообщения от каждого процесса каждому (см. рис. 4). В качестве передаваемого сообщения используйте номер процесса. Каждый процесс должен вывести на экран все полученные сообщения.



1. : передача к 0
2. : передача к 2
3. : прием от 0
4. : прием от 2

 - Процесс с номером X

[t] - Последовательность выполнения действия, где t – порядковый номер

Входные данные: нет

Выходные данные: каждый процесс выводит сообщение "[номер_процесса]: receive message '<сообщение>' from <номер_процесса>"

| Входные данные | Выходные данные (3 процесса) |
|----------------|------------------------------|
|----------------|------------------------------|

| | |
|--|---------------------------------|
| | [0]: receive message '1' from 1 |
| | 1. : receive message '2' from 2 |
| | 2. : receive message '0' from 0 |
| | 1. : receive message '2' from 2 |
| | 2. : receive message '0' from 0 |
| | [2]: receive message '1' from 1 |

Полный текст программы:

1. Класс Main

```
var lab8 = new Lab8("Lab8", args);  
lab8.Start();
```

2. Класс Lab8

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using MPI;
using ProgramMPI.Interfaces;

namespace ProgramMPI.Labs
{
    public class Lab8 : ILab
    {
        public string Name { get; set; }
        public string[] _args { get; set; }

        public Lab8(string name, string[] args)
        {
            Name = name;
            _args = args;
        }

        public void ProcessStart(string[] args)
        {
            using (new MPI.Environment(ref args))
            {
                Intracommunicator comm = Communicator.world;

                int rank = comm.Rank;
                int size = comm.Size;

                for (int i = 0; i < size; i++)
                {
                    if (i != rank)
                    {
                        // Отправка сообщения от текущего процесса к процессу i
                        comm.Send(rank, i, 0);
                    }
                }

                for (int i = 0; i < size - 1; i++)
                {
                    // Получение сообщения от любого процесса
                    int receivedRank = comm.Receive<int>(Communicator.anySource, 0);

                    Print(comm.Rank, receivedRank);
                }
            }
        }

        public void Start()
        {
            Console.WriteLine(this);

            ProcessStart(_args);
        }
    }
}
```

```
        Console.WriteLine("\n");
    }

    public override string ToString() => $"{Name} started classes:";

    private void Print(int rank, int receivedMessage) =>

        Console.WriteLine($"{rank}: receive message '{receivedMessage}' from  
{receivedMessage}");

    }
}
```

Результат:

```
Lab8 started classes:
[2]: receive message '0' from 0
[0]: receive message '2' from 2
[2]: receive message '1' from 1
[0]: receive message '1' from 1
[1]: receive message '0' from 0
[1]: receive message '2' from 2
```


Задание 23. Коллективные коммуникации (широковещательная рассылка данных):

Задание для выполнения

Изучите MPI-функцию широковещательной рассылки данных MPI_Bcast. Напишите MPI-программу, которая в строке длины n определяет количество вхождений символов. Ввод данных должен осуществляться процессом с номером 0. Для рассылки строки поиска и ее длины по процессам используйте функцию MPI_Bcast.

Перепишите программу, используя вместо функции MPI_Bcast функции коммуникации «точка-точка». Сравните эффективность выполнения программ с коллективными и точечными обменами.

Входные данные: целое число n ($1 \leq n \leq 100$), строка из n символов (каждый символ в строке может представлять собой только строчную букву английского алфавита)

Выходные данные: "<буква> = <сообщение>"

| Входные данные | Выходные данные |
|----------------|--|
| 9 aaaaaaaaa | a = 9 |
| 3 rvtabc | a = 1 b = 1 c = 1 r = 1 t = 1 v = 1 |

Полный текст программы:

1. Класс Main

```
var lab9 = new Lab9("Lab9", args);  
lab9.Start();  
  
var lab9 = new Lab9("Lab9", args, true);  
lab9.Start();
```

2. Класс Lab9

```
using MPI;
using ProgramMPI.Interfaces;

namespace ProgramMPI.Labs
{
    public class Lab9 : ILab
    {
        public string Name { get; set; }
        private string[] _args { get; set; }
        private bool _isBCast { get; set; }

        public Lab9(string name, string[] args, bool isBCast = false)
        {
            Name = name;
            _args = args;
            _isBCast = isBCast;
        }

        public void ProccessStart(string[] args)
        {
            using (new MPI.Environment(ref args))
            {
                Intracommunicator comm = Communicator.world;

                int root = 0;
                string inputString = null;

                if (comm.Rank == root)
                {
                    Console.WriteLine("Введите строку:");
                    inputString = Console.ReadLine();

                    // Отправка длины строки каждому процессу
                    for (int i = 1; i < comm.Size; i++)
                    {
                        comm.Send(inputString.Length, i, 0);
                    }
                }

                int stringLength = 0;

                if (comm.Rank != root)
                {
                    // Получение длины строки
                    stringLength = comm.Receive<int>(root, 0);
                }
                else
                {
                    stringLength = inputString.Length;
                }

                char[] charArray = new char[stringLength];

                if (comm.Rank == root)
```

```

    {
        charArray = inputString.ToCharArray();

        // Отправка строки каждому процессу
        for (int i = 1; i < comm.Size; i++)
        {
            comm.Send(charArray, i, 0);
        }
    }
    else
    {
        // Получение строки
        charArray = comm.Receive<char[]>(root, 0);
    }

    // Подсчет символов
    Dictionary<char, int> charCount = new Dictionary<char, int>();
    foreach (char c in charArray)
    {
        if (charCount.ContainsKey(c))
        {
            charCount[c]++;
        }
        else
        {
            charCount.Add(c, 1);
        }
    }

    // Вывод результатов
    foreach (var entry in charCount)
    {
        Print(entry.Key, entry.Value);
    }
}

private void ProccessStartBCast(string[] args)
{
    using (new MPI.Environment(ref args))
    {
        Intracommunicator comm = Communicator.world;

        int root = 0; // Процесс, осуществляющий ввод и начальную рассылку

        string inputString = null;

        if (comm.Rank == root)
        {
            Console.WriteLine("Введите строку:");
            inputString = Console.ReadLine(); // Ввод строки процессом 0
        }

        // Рассылка длины строки всем процессам
        int stringLength = inputString?.Length ?? 0;
        comm.Broadcast(ref stringLength, root);
    }
}

```

данных

```

// Инициализация массива символов для приема строки
char[] charArray = new char[stringLength];

if (comm.Rank == root)
{
    charArray = inputString.ToCharArray();
}

// Рассылка самой строки всем процессам
comm.Broadcast(ref charArray, root);

// Подсчет символов
Dictionary<char, int> charCount = new Dictionary<char, int>();
foreach (char c in charArray)
{
    if (charCount.ContainsKey(c))
    {
        charCount[c]++;
    }
    else
    {
        charCount.Add(c, 1);
    }
}

// Вывод результатов
foreach (var entry in charCount)
{
    Print(entry.Key, entry.Value);
}
}

public void Start()
{
    Console.WriteLine(this);

    if (!_isBCast)
        ProccessStart(_args);
    else
        ProccessStartBCast(_args);

    Console.WriteLine("\n");
}

public override string ToString() => $"{Name} started classes:";

private void Print(char key, int value) =>
    Console.WriteLine($"{key} = {value}");
}
}

```

Результат:

```
Lab9 started classes:  
Введите строку:  
aaaaaaaaaa  
a = 9
```

```
Lab9 started classes:  
Введите строку:  
rvtabc  
r = 1  
v = 1  
t = 1  
a = 1  
b = 1  
c = 1
```

Задание 24. Коллективные коммуникации (операции редукции):

Задание для выполнения

Изучите MPI-функцию для выполнения операций редукции над данными, расположенными в адресных пространствах различных процессов, MPI_Reduce. Реализуйте программу вычисления числа π (см. задание 8), используя функцию MPI_Reduce для суммирования результатов, вычисленных каждым процессом.

Перепишите программу, используя вместо функции MPI_Reduce функции коммуникации «точка-точка». Сравните эффективность выполнения программ с коллективными и точечными обменами.

Входные данные: одно целое число N (точность вычисления)

Выходные данные: одно вещественное число π .

| Входные данные | Выходные данные |
|----------------|-----------------|
| 1000000000 | 3.14159265 |

Полный текст программы:

1. Класс Main

```
var lab10 = new Lab10("Lab10", args);  
lab10.Start();
```

2. Класс Lab10

```
using System.Diagnostics;
using MPI;
using ProgramMPI.Interfaces;

namespace ProgramMPI.Labs
{
    public class Lab10 : ILab
    {
        public string Name { get; set; }
        private string[] _args { get; set; }
        private bool _isBCast { get; set; }
        private Stopwatch _stopwatch = new Stopwatch();

        public Lab10(string name, string[] args, bool isBCast = false)
        {
            Name = name;
            _args = args;
            _isBCast = isBCast;
        }

        public void ProccessStart(string[] args)
        {
            //Console.WriteLine("Введите целое число N: ");
            //if (!int.TryParse(Console.ReadLine(), out int N) && N <= 0)
            //{
            //    Console.WriteLine("Некорректный ввод. Убедитесь, что вводите
положительное целое число.");
            //    return;
            //}

            using (new MPI.Environment(ref args))
            {
                _stopwatch.Start();

                Intracommunicator comm = Communicator.world;
                int n = 1000000, localN;
                double step, sum = 0.0, pi;
                step = 1.0 / n;
                localN = n / comm.Size;

                for (int i = comm.Rank * localN; i < (comm.Rank + 1) * localN; i++)
                {
                    double x = (i + 0.5) * step;
                    sum += 4.0 / (1.0 + x * x);
                }

                double localSum = sum * step;

                // Использование MPI_Reduce для редукции результатов
                double globalSum = comm.Reduce(localSum, Operation<double>.Add, 0);

                _stopwatch.Stop();

                if (comm.Rank == 0)
```

```

        {
            pi = globalSum;
            Print(pi, _stopwatch.Elapsed);
        }
    }
}

private void ProccessStartBCast(string[] args)
{
    using (new MPI.Environment(ref args))
    {
        _stopwatch.Start();

        Intracommunicator comm = Communicator.world;
        int n = 1000000, localN;
        double step, localSum = 0.0, pi = 0.0;

        step = 1.0 / n;
        localN = n / comm.Size;

        // Вычисление локальной суммы для каждого процесса
        for (int i = comm.Rank * localN; i < (comm.Rank + 1) * localN; i++)
        {
            double x = (i + 0.5) * step;
            localSum += 4.0 / (1.0 + x * x);
        }
        localSum *= step;

        if (comm.Rank == 0)
        {
            // Процесс с рангом 0 суммирует свою локальную сумму и получает
            значения от других процессов
            pi = localSum;
            for (int i = 1; i < comm.Size; i++)
            {
                double tempSum = comm.Receive<double>(i, 0);
                pi += tempSum;
            }

            _stopwatch.Stop();

            Print(pi, _stopwatch.Elapsed);
        }
        else
        {
            // Отправка локальной суммы процессу с рангом 0
            comm.Send(localSum, 0, 0);
        }
    }
}

public void Start()
{
    Console.WriteLine(this);

    if (!_isBCast)
        ProccessStart(_args);
}

```



```
        else
            ProccessStartBCast(_args);

        Console.WriteLine("\n");
    }

    public override string ToString() => $"{Name} started classes:";

    private void Print(double pi, TimeSpan time) => Console.WriteLine($"Result PI
= {pi}, Time = {time.Milliseconds} Mc");

    }
}
```

Результат:

```
Lab10 started classes:
Result PI = 3,1415926535898993, Time = 260 Mc
```

```
Lab10 started classes:
Result PI = 3,1415926535898993, Time = 260 Mc
```

Задание 25. Коллективные коммуникации (функции распределения и сбора данных):

Задание для выполнения

Изучите MPI-функции распределения и сбора блоков данных по процессам `MPI_Scatter` и `MPI_Gather`. Напишите программу, которая вычисляет произведение двух квадратных матриц $A \times B = C$ размера $n \times n$. Используйте формулу, приведенную в задании 9. Ввод данных и вывод результата должны осуществляться процессом с номером 0. Для распределения матриц `A` и `B` и сбора матрицы `C` используйте функций `MPI_Scatter` и `MPI_Gather`.

Перепишите программу, используя вместо функций `MPI_Scatter` и `MPI_Gather` функции коммуникации «точка-точка». Сравните эффективность выполнения программ с коллективными и точечными обменами.

Входные данные: целое число n , $1 \leq n \leq 10$, n_2 вещественных элементов матрицы `A` и n_2 вещественных элементов матрицы `B`

Выходные данные: n_2 вещественных элементов матрицы `C`

| Входные данные | Выходные данные |
|----------------|-----------------|
| 2 | 14 4 |
| 1 3 | 44 16 |
| 3 8 | |
| 5 4 | |
| 3 0 | |

Полный текст программы:

1. Класс Main

```
var lab11 = new Lab11("Lab11", args);
lab11.Start();

lab11 = new Lab11("Lab11", args, true);
lab11.Start();
```

2. Класс Lab11

Результат:

Задание 26. Группы и коммутаторы:

Задание для выполнения

Напишите программу, в которой производится широковещательная рассылка сообщения с помощью функции `MPI_Bcast`, но только по процессам с четным номером. Для рассылки сообщения создайте новый коммутатор.

Каждый процесс приложения должен выводить на экран «MPI_COMM_WORLD:

"<номер процесса в коммутаторе MPI_COMM_WORLD> from <количество процессов в коммутаторе MPI_COMM_WORLD>. New comm: <номер процесса в новом коммутаторе> from <количество процессов в новом коммутаторе>. Message = <сообщение>"

Входные данные: message – строка с сообщением, считываемым только процессом 0, количество символов в message от 1 до 10

Выходные данные: строки вида "MPI_COMM_WORLD: <номер процесса в коммутаторе MPI_COMM_WORLD> from <количество процессов в коммутаторе MPI_COMM_WORLD>. New comm: <номер процесса в новом коммутаторе> from <количество процессов в новом коммутаторе>. Message = <сообщение>"

| Входные данные | Выходные данные |
|----------------|-----------------|
|----------------|-----------------|

| | |
|---|--|
| A | MPI_COMM_WORLD: 0 from 3. New comm: 0 from 2. Message = A |
| | MPI_COMM_WORLD: 1 from 3. New comm: no from no. Message = no |
| | MPI_COMM_WORLD: 2 from 3. New comm: 1 from 2. Message = A |

Полный текст программы:

1. Класс Main

```
var lab12 = new Lab12("Lab12", args);
lab12.Start();

lab12 = new Lab12("Lab12", args, true);
lab12.Start();
```

2. Класс Lab12

Результат: