Лабораторная работа №1

Передача и прием сообщений в MPI

Максим Ромашкин

ПИН-34   Номер в списке: 18

Оглавление

[**Задание, код программы и пояснение:** 1](#_Toc95838769)

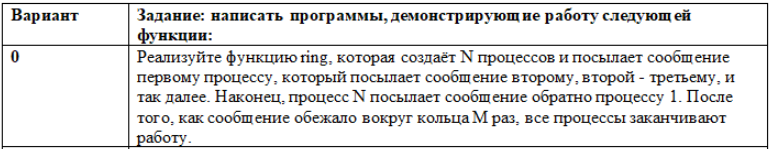
[**Листинг программы:** 1](#_Toc95838770)

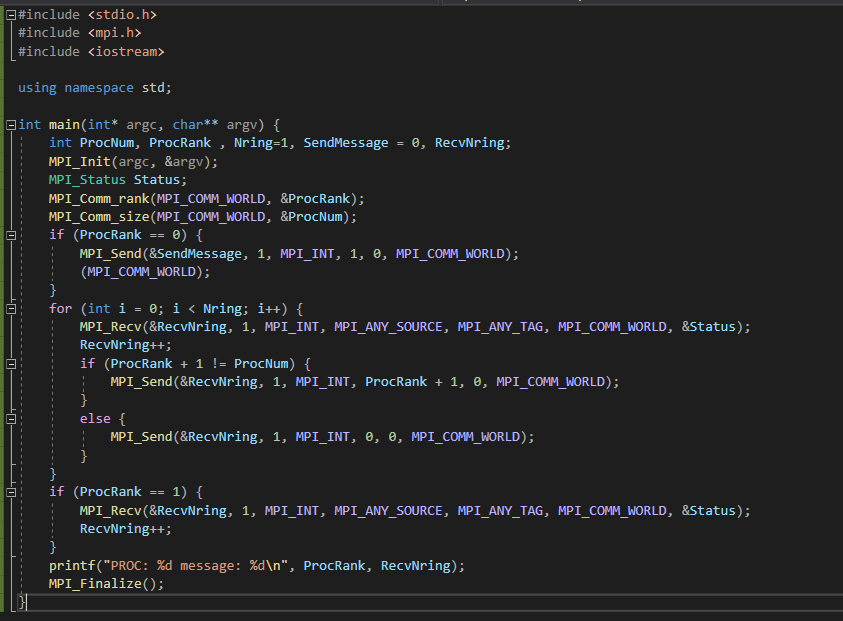
[**Результат программы:** 1](#_Toc95838771)

[**Выполнение примера** 1](#_Toc95838772)

[**Ответы на контрольные вопросы** 1](#_Toc95838773)

# **Задание, код программы и пояснение:**

  
В коде сначала отправляется сообщение (SendMessage = 0) от 0 процесса к 1. Все процессы входят в цикл и начинают ожидать передачи сообщения. Процесс номер 1, получает сообщение увеличивает его на 1 и отправляет процессу с номером 2 и т.д. По сути, после того как каждый процесс получит N сообщений цикл закончится. На последний итерации сообщение получит 0 процесс и отправит сообщение 1, после чего выйдет из цикла. После этого первым циклом мы принимаем сообщение.



# **Листинг программы:**

#include <stdio.h>

#include <mpi.h>

#include <iostream>

using namespace std;

int main(int\* argc, char\*\* argv) {

int ProcNum, ProcRank , Nring=1, SendMessage = 0, RecvNring;

MPI\_Init(argc, &argv);

MPI\_Status Status;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &ProcRank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &ProcNum);

if (ProcRank == 0) {

MPI\_Send(&SendMessage, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD);

(MPI\_COMM\_WORLD);

}

for (int i = 0; i < Nring; i++) {

MPI\_Recv(&RecvNring, 1, MPI\_INT, MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &Status);

RecvNring++;

if (ProcRank + 1 != ProcNum) {

MPI\_Send(&RecvNring, 1, MPI\_INT, ProcRank + 1, 0, MPI\_COMM\_WORLD);

}

else {

MPI\_Send(&RecvNring, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

}

}

if (ProcRank == 1) {

MPI\_Recv(&RecvNring, 1, MPI\_INT, MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &Status);

RecvNring++;

}

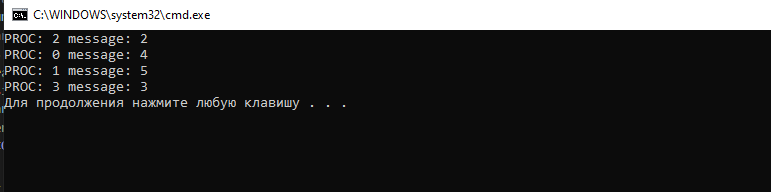
printf("PROC: %d message: %d\n", ProcRank, RecvNring);

MPI\_Finalize();

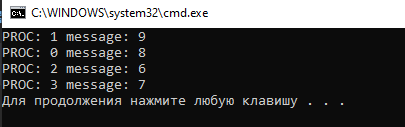
}

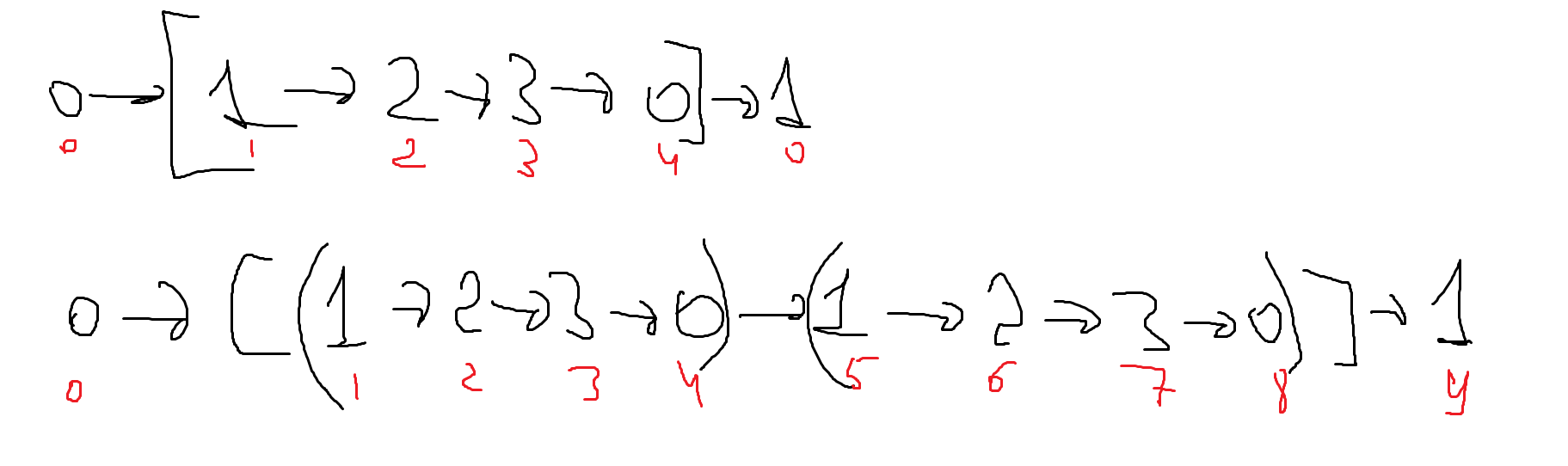
# **Результат программы:**

Результат для 1 круга:

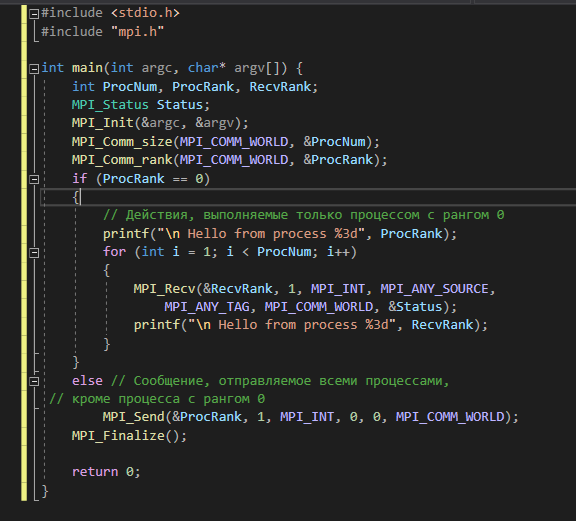


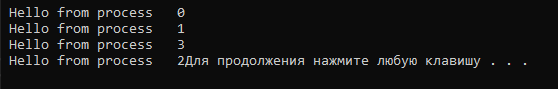
Результат для 2 кругов:



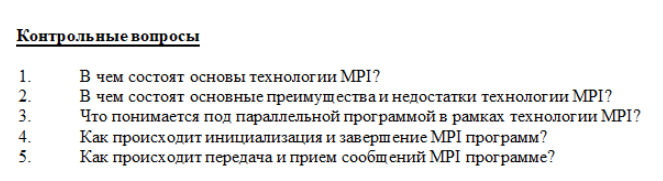
Черные цифры — номера процессов, красные — какое значение сообщения они хранят. В квадратных скобках цикл.  


# **Выполнение примера**





# **Ответы на контрольные вопросы**



1. Основу MPI составляют операции передачи сообщений. Среди предусмотренных в составе MPI функций различаются парные (point-to-point) операции между двумя процессами и коллективные (collective) коммуникационные действия для одновременного взаимодействия нескольких процессов.
2. **Преимущества**:
   1. MPI помогает решить проблему переносимости параллельных программ между разными компьютерными системами.
   2. MPI содействует повышению эффективности параллельных вычислений: практически для каждого типа вычислительных систем существуют реализации библиотек MPI, учитывающие возможности используемого коммуникационного оборудования.
   3. MPI облегчает процесс написания параллельных программ, когда при разработке используются библиотеки программных модулей, написанных с использованием этого интерфейса.
   4. MPI позволяет создавать хорошо масштабируемые параллельные программы.

**Недостатки**:

1. MPI является низкоуровневым инструментом программиста.
2. Не существует реализаций MPI, в полной мере обеспечивающих совмещение обменов с вычислениями.
3. MPI не предоставляет механизмов задания начального размещения процессов по процессорам.
4. Полномасштабная отладка MPI-программ затруднительна вследствие одновременного исполнения нескольких программных ветвей.
5. Под параллельной программой в рамках MPI понимается множество одновременно выполняемых процессов. Процессы могут выполняться как на разных процессорах, так и на одном.
6. Первой вызываемой функцией MPI должна быть функция:  
     
   int **MPI\_Init**(int \*agrc, char \*\*\*argv);  
     
   для инициализации среды выполнения MPI-программы. Параметрами функции являются количество аргументов в командной строке и текст самой командной строки.  
   Последней вызываемой функцией MPI обязательно должна являться функция:  
     
   int **MPI\_Finalize**(void);
7. Для **передачи** сообщения процесс-отправитель должен выполнить функцию:  
      
   int **MPI\_Send**(void \*buf, int count, MPI\_Datatype type, int dest,int tag, MPI\_Comm comm);   
     
   В момент завершения функции **MPI\_Send** состояние самого пересылаемого сообщения может быть совершенно различным - сообщение может:  
   - располагаться в процессе-отправителе,  
   - находиться в процессе передачи,  
   - храниться в процессе-получателе  
   - или же может быть принято процессом-получателем при помощи функции **MPI\_Recv**.  
   Завершение функции MPI\_Send означает лишь, что операция передачи начала выполняться.  
     
   Для **приема** сообщения процесс-получатель должен выполнить функцию:  
     
   int **MPI\_Recv**(void \*buf, int count, MPI\_Datatype type, int source,int tag, MPI\_Comm comm, MPI\_Status \*status);  
     
   Вызов функции **MPI\_Recv** не должен согласовываться со временем вызова соответствующей функции передачи сообщения **MPI\_Send** – прием сообщения может быть инициирован до момента, в момент или после момента начала отправки сообщения.  
     
   По завершении функции **MPI\_Recv** в заданном буфере памяти будет располагаться принятое сообщение. Принципиальный момент здесь состоит в том, что функция **MPI\_Recv** является блокирующей для процесса-получателя, т.е. его выполнение приостанавливается до завершения работы функции. Таким образом, если ожидаемое для приема сообщение будет отсутствовать, выполнение параллельной программы будет блокировано.