**Лабораторная работа №2**

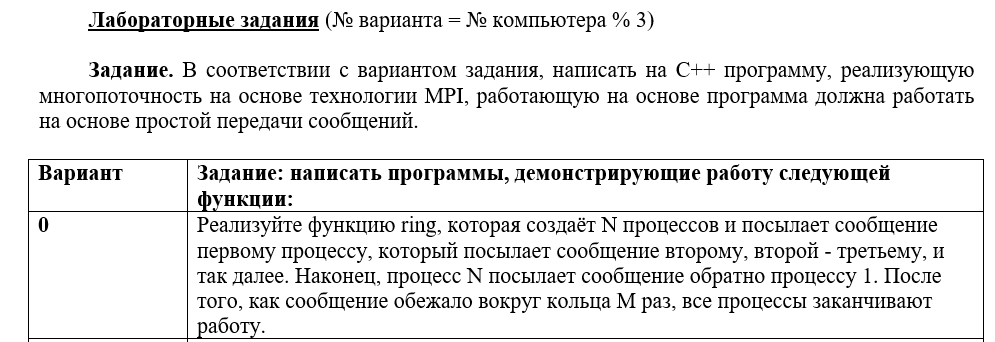
**Макеева Настя ПИН-31**

**Коллективные операции передачи данных**

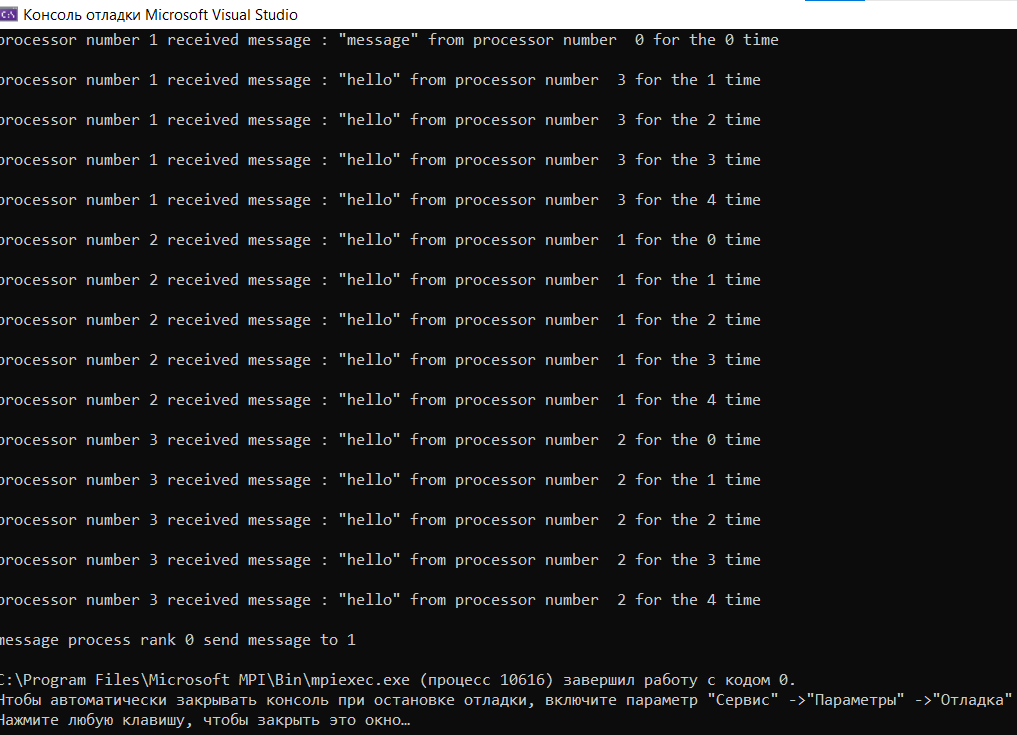
**Цель:** изучить основные принципы коллективных операций передачи данных в технологии MPI на примере использования в рамках языка С++.

**Лабораторные задания**

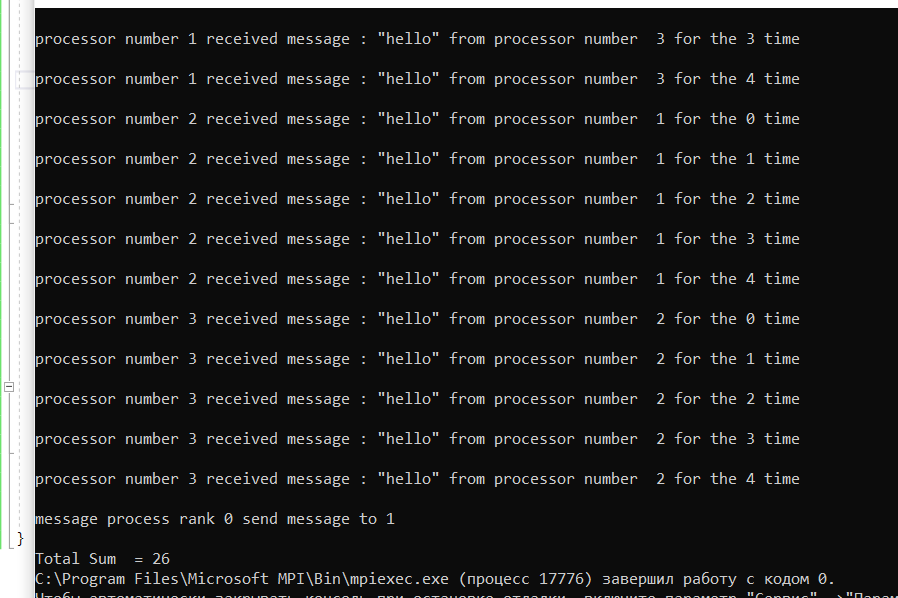
**Задание.** Модифицировать программу, написанную на Л.Р. №1, так чтобы она работала на основе коллективной передачи сообщений. **Результаты работы сравнить и занести в отчет.**



Lab1



Lab2



#include <string>

#include <vector>

#include <cstdlib>

#include <iostream>

#include <stdio.h>

#include "mpi.h"

#include "stdlib.h"

int main(int argc, char\* argv[])

{

int ProcNum, ProcRank, RingNum;

//RingNum = atoi(argv[1]);

RingNum = 5;

int recvFrom;

int sendTo;

char hello[] = "hello";

//MPI\_Status status; //стату возврата

MPI\_Init(&argc, &argv);

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

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &ProcRank); //возвращает ранг процесса. 1ый аргумент - коммуникатор

if (ProcRank == 0)

{

char message[] = "message";

MPI\_Send(message, strlen(message) + 1, MPI\_CHAR, 1, 0, MPI\_COMM\_WORLD);

printf("%s process rank %d send message to 1 \n", message, ProcRank);

}

else

{

for (int i = 0; i < RingNum; i++)

{

char message[30];

if (i > 0 && ProcRank == 1) { recvFrom = ProcNum - 1; }

else { recvFrom = ProcRank - 1; }

MPI\_Recv(message, strlen(message) + 1, MPI\_CHAR, recvFrom, 0, MPI\_COMM\_WORLD, MPI\_STATUSES\_IGNORE);

printf("processor number %d received message : \"%s\" from processor number %d for the %d time \n\n", ProcRank, message, recvFrom, i);

if (ProcRank == ProcNum - 1) sendTo = 1;

else sendTo = ProcRank + 1;

//отправляем сообщения

MPI\_Send(hello, strlen(hello) + 1, MPI\_CHAR, sendTo, 0, MPI\_COMM\_WORLD);

}

}

int TotalSum = 0;

int ProcSum = ProcRank + RingNum;

// сборка частичных сумм на процессе с рангом 0

MPI\_Reduce(&ProcSum, &TotalSum, 1, MPI\_INT, MPI\_SUM, 0, MPI\_COMM\_WORLD);

// ждем все процессы, чтоб просуммировать все значения

MPI\_Barrier(MPI\_COMM\_WORLD);

if (ProcRank == 0)

{

printf( "\nTotal Sum = %d", TotalSum );

}

MPI\_Finalize();

return 0;

**Контрольные вопросы**

1. Как происходит передача данных от одного процесса всем?

Ш**ироковещательная рассылка** данных может быть обеспечено при помощи функции MPI:

int MPI\_Bcast(void \*buf,int count,MPI\_Datatype type,int root,MPI\_Comm comm);

где buf, count, type – буфер памяти с отправляемым сообщением (для процесса с рангом 0),и для приема сообщений для всех остальных процессов,

root - ранг процесса, выполняющего рассылку данных,

comm - коммуникатор, в рамках которого выполняется передача данных.

Функция **MPI\_Bcast** осуществляет рассылку данных из буфера **buf**, содержащего count элементов типа **type** с процесса, имеющего номер **root**, всем процессам, входящим в коммуникатор **comm**.

- функция **MPI\_Bcast** определяет коллективную операцию и, тем самым, при выполнении необходимых рассылок данных вызов функции **MPI\_Bcast** должен быть осуществлен всеми процессами указываемого коммуникатора;

- указываемый в функции **MPI\_Bcast** буфер памяти имеет различное назначение в разных процессах. Для процесса с рангом **root**, с которого осуществляется рассылка данных, в этом буфере должно находиться рассылаемое сообщение. Для всех остальных процессов указываемый буфер предназначен для приема передаваемых данных.

1. Как происходит передача данных от всем процессов одному?

Операция редукции:

int MPI\_Reduce(void \*sendbuf, void \*recvbuf,int count,MPI\_Datatype type,

MPI\_Op op,int root,MPI\_Comm comm);

1. Какие используются в MPI для синхронизации вычислений?

**Синхронизация** процессов, т.е. одновременное достижение процессами тех или иных точек процесса вычислений, обеспечивается при помощи функции MPI:

int MPI\_Barrier(MPI\_Comm comm);

Функция **MPI\_Barrier** должна вызываться всеми процессами используемого коммуникатора. При вызове функции **MPI\_Barrier** выполнение процесса блокируется, продолжение вычислений процесса происходит только после вызова функции **MPI\_Barrier** всеми процессами коммуникатора.

1. Как организуется неблокирующий обмен данными между процессами?

Наименование неблокирующих аналогов образуется из названий соответствующих функций путем добавления префикса **I** (**Immediate**). Список параметров неблокирующих функций содержит весь набор параметров исходных функций и один дополнительный параметр **request** с типом **MPI\_Request** (в функции **MPI\_Irecv** отсутствует также параметр **status**):

int MPI\_Isend(void \*buf, int count, MPI\_Datatype type, int dest,

int tag, MPI\_Comm comm, MPI\_Request \*request);

int MPI\_Issend(void \*buf, int count, MPI\_Datatype type, int dest,

int tag, MPI\_Comm comm, MPI\_Request \*request);

int MPI\_Ibsend(void \*buf, int count, MPI\_Datatype type, int dest,

int tag, MPI\_Comm comm, MPI\_Request \*request);

int MPI\_Irsend(void \*buf, int count, MPI\_Datatype type, int dest,

int tag, MPI\_Comm comm, MPI\_Request \*request);

int MPI\_Irecv(void \*buf, int count, MPI\_Datatype type, int source,

int tag, MPI\_Comm comm, MPI\_Request \*request);

1. Как организуется одновременное выполнение прием и передачи данных?

Достижение эффективного и гарантированного одновременного выполнения операций передачи и приема данных может быть обеспечено при помощи функции MPI:

int MPI\_Sendrecv(void \*sbuf,int scount,MPI\_Datatype stype,int dest, int stag,

void \*rbuf,int rcount,MPI\_Datatype rtype,int source,int rtag,

MPI\_Comm comm, MPI\_Status \*status);

Где:

sbuf, scount, stype, dest, stag - параметры передаваемого сообщения;

rbuf, rcount, rtype, source, rtag - параметры принимаемого сообщения;

comm - коммуникатор, в рамках которого выполняется передача данных;

status – структура данных с информацией о результате выполнения операции.

Функция MPI\_Sendrecv передает сообщение, описываемое параметрами (**sbuf**, **scount**, **stype**, **dest**, **stag**), процессу с рангом **dest** и принимает сообщение в буфер, определяемый параметрами (**rbuf, rcount, rtype**, **source**, **rtag**), от процесса с рангом **source**.

В функции **MPI\_Sendrecv** для передачи и приема сообщений применяются разные буфера. В случае же, когда сообщения имеют одинаковый тип, в MPI имеется возможность использования единого буфера:

int MPI\_Sendrecv\_replace (void \*buf, int count, MPI\_Datatype type, int dest,

int stag, int source, int rtag, MPI\_Comm comm, MPI\_Status\* status)