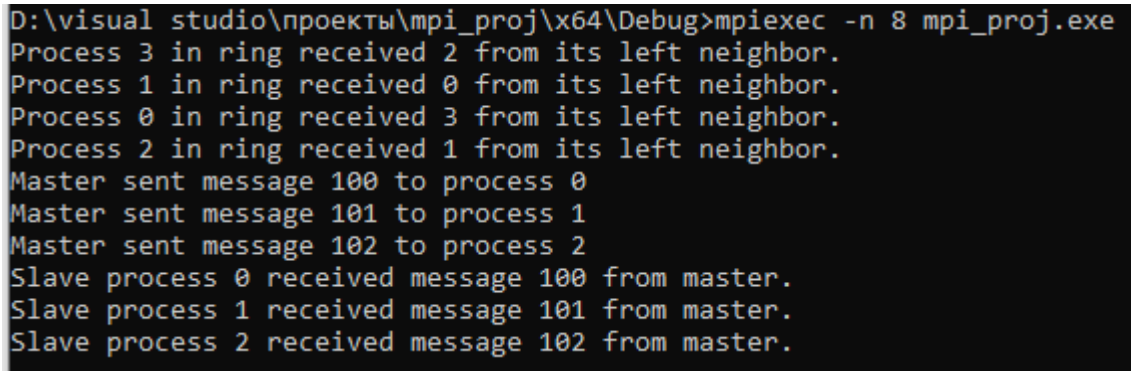
Лабораторная работа №12

Виртуальные топологии

Код: https://pastebin.com/DD5Drz7R



Приложение. Код.

```
#include <mpi.h>

#include <iostream>

using namespace std;

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);

    int rank, size;
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if (size % 2 != 0) {
        if (rank == 0) {
            cerr << "Error: Number of processes must be even." << endl;
        }
        MPI_Finalize();
        return 1;
    }

    // создание коммуникаторов для подгрупп
    int ring_size = size / 2;
    int master_slave_size = size / 2;

    MPI_Comm ring_comm;
    MPI_Comm master_slave_comm;

    // разделение процессов
    MPI_Comm_split(MPI_COMM_WORLD, rank < ring_size ? 0 : 1, rank, & ring_comm);
    MPI_Comm_split(MPI_COMM_WORLD, rank >= ring_size ? 0 : 1,
        rank % master_slave_size, &master_slave_comm);

    // создание декартовой топологии
    int dims[1] = { ring_size };  // Одномерная топология
    int periods[1] = { 1 };       // Периодичность для кольца
```

```
MPI_Cart_create(MPI_COMM_WORLD, 1, dims, periods, 0, &ring_comm);

// Кольцевой обмен
if (rank < ring_size) {
    int send_data = rank;
    int recv_data;

    // получаем смещенные исходные и целевые ранги
    int left_neighbor, right_neighbor;
    MPI_Cart_shift(ring_comm, 0, 1, &left_neighbor, &right_neighbor);

    MPI_Sendrecv(&send_data, 1, MPI_INT, right_neighbor, 0, &recv_data, 1,
        MPI_INT, left_neighbor, 0, ring_comm, MPI_STATUS_IGNORE);
    cout << "Process " << rank << " in ring received " << recv_data
        << " from its left neighbor." << endl;
}

// master-slave обмен
if (rank >= ring_size) {
    MPI_Comm_rank(master_slave_comm, &rank);
    MPI_Comm StarComm;
    int index[] = { 1, 2, 3, 6 };
    int edges[] = { 3, 3, 3, 0, 1, 2 };

    // создание графовой топологии
    MPI_Graph_create(master_slave_comm, 4, index, edges, 0, &StarComm);

    // получение соседей для текущего процесса
    int max_neighbors =
        3;  // максимальное количество соседей для текущего процесса
    int* neighbor_ranks = (int*)malloc(max_neighbors * sizeof(int));
    int curr_neighbors;

    MPI_Graph_neighbors_count(StarComm, rank, &curr_neighbors);
    MPI_Graph_neighbors(StarComm, rank, max_neighbors, neighbor_ranks);

    // рассылаем сообщения в master процессе
    if (rank == 3) {
```

```
            for (int i = 0; i < curr_neighbors; i++) {
                int message = i + 100;
                MPI_Send(&message, 1, MPI_INT, neighbor_ranks[i], 0, StarComm);
                cout << "Master sent message " << message << " to process "
                    << neighbor_ranks[i] << endl;
            }
        }
        // принимаем сообщения в slave процессах
        else {
            int received_message;
            MPI_Recv(&received_message, 1, MPI_INT, 3, MPI_ANY_TAG, StarComm,
                MPI_STATUS_IGNORE);
            cout << "Slave process " << rank << " received message "
                << received_message << " from master." << endl;
        }
        MPI_Comm_free(&master_slave_comm);
        free(neighbor_ranks);
    }

    MPI_Finalize();
    return 0;
}
```