

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школы электроники и компьютерных наук
Кафедра системного программирования

ОТЧЕТ
о лабораторной работе №5
по дисциплине «Технологии параллельного программирования»

Выполнил:
студент группы КЭ-220
_____/Голенищев А. Б.
_____ 2024 г.

Отчет принял:
_____/Жулев А. Э.
_____ 2024 г.

Челябинск
2024

Задание 13. Коммуникации «точка-точка»: схема «эстафетная палочка»

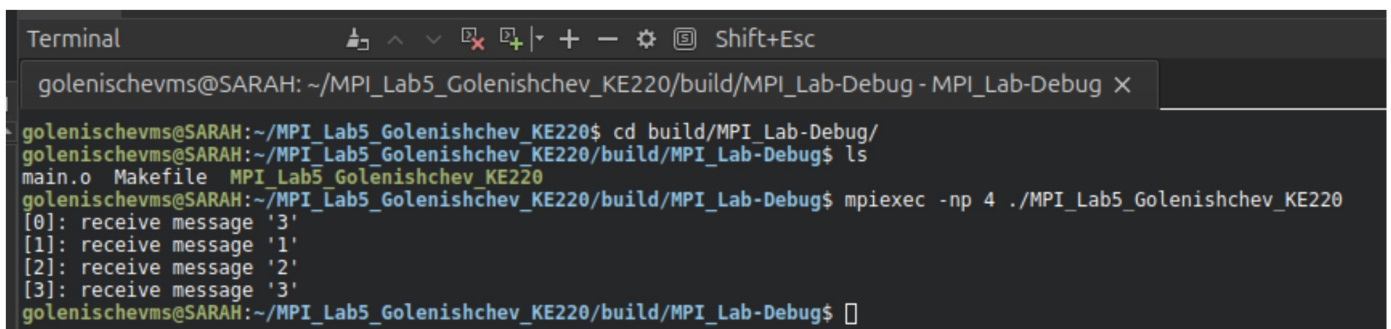
Разработали программу обмена сообщениями между процессами в эстафетном порядке, листнинг 1. Представлен результат ее работы, рисунок 1.

```
#include <mpi.h>
#include <stdio.h>
// Golenishchev Artem, KE-220 Task 13
int main(int argc, char** argv) {
    int rank, size, buf;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); // Получаем номер текущего процесса
    MPI_Comm_size(MPI_COMM_WORLD, &size); // Получаем общее количество процессов

    if (rank == 0) {
        // Процесс 0 начинает передачу
        buf = 0;
        MPI_Send(&buf, 1, MPI_INT, 1, 0, MPI_COMM_WORLD); // Отправляем процессу 1
        MPI_Recv(&buf, 1, MPI_INT, size - 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE); //
        Принимаем от последнего процесса
    } else {
        // Остальные процессы
        MPI_Recv(&buf, 1, MPI_INT, rank - 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE); /
        / Ждём сообщение от предыдущего процесса
        MPI_Send(&(++buf), 1, MPI_INT, (rank + 1) % size, 0, MPI_COMM_WORLD);
        // Увеличиваем и отправляем следующему
    }
    // Вывод сообщения
    printf("[%d]: receive message '%d'\n", rank, buf);
    MPI_Finalize();
    return 0;
}
```

Листнинг 1. Код программы обмена сообщений между процессами



```
Terminal
golenishchevms@SARAH: ~/MPI_Lab5_Golenishchev_KE220/build/MPI_Lab-Debug - MPI_Lab-Debug x
golenishchevms@SARAH:~/MPI_Lab5_Golenishchev_KE220$ cd build/MPI_Lab-Debug/
golenishchevms@SARAH:~/MPI_Lab5_Golenishchev_KE220/build/MPI_Lab-Debug$ ls
main.o Makefile MPI_Lab5_Golenishchev_KE220
golenishchevms@SARAH:~/MPI_Lab5_Golenishchev_KE220/build/MPI_Lab-Debug$ mpirun -np 4 ./MPI_Lab5_Golenishchev_KE220
[0]: receive message '3'
[1]: receive message '1'
[2]: receive message '2'
[3]: receive message '3'
golenishchevms@SARAH:~/MPI_Lab5_Golenishchev_KE220/build/MPI_Lab-Debug$
```

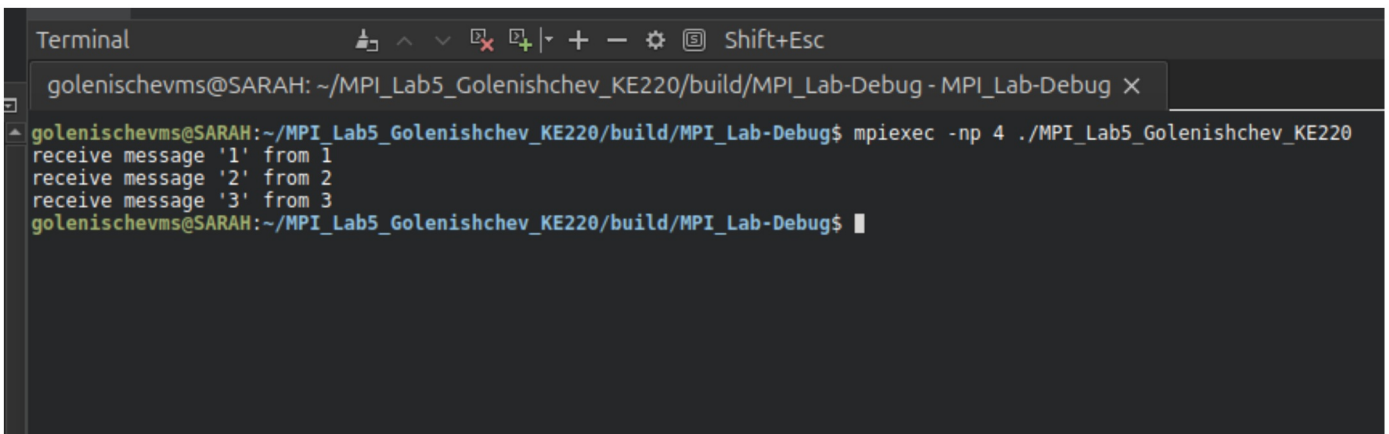
Рисунок 1. Результат работы программы обмена сообщений между процессами

Задание 14. Коммуникации «точка-точка»: схема «мастер-рабочие»

Разработали программу обмена сообщениями между процессами от многих одному главному, листнинг 2. Представлен результат ее работы, рисунок 2.

```
#include <mpi.h>
#include <stdio.h>
// Golenishchev Artem, KE-220 Task 14
int main(int argc, char** argv) {
    int rank, size, buf;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); // Получение номера текущего процесса
    MPI_Comm_size(MPI_COMM_WORLD, &size); // Получение общего числа процессов
    if (rank == 0) {
        // Код для master-процесса
        for (int src = 1; src < size; src++) {
            MPI_Recv(&buf, 1, MPI_INT, src, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            printf("receive message '%d' from %d\n", buf, src);
        }
    } else {
        // Код для slave-процессов
        buf = rank; // Номер процесса
        MPI_Send(&buf, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    }
    MPI_Finalize();
    return 0;
}
```

Листнинг 2. Код программы «мастер-рабочие»



```
Terminal
golenishevms@SARAH: ~/MPI_Lab5_Golenishchev_KE220/build/MPI_Lab-Debug - MPI_Lab-Debug x
golenishevms@SARAH:~/MPI_Lab5_Golenishchev_KE220/build/MPI_Lab-Debug$ mpirun -np 4 ./MPI_Lab5_Golenishchev_KE220
receive message '1' from 1
receive message '2' from 2
receive message '3' from 3
golenishevms@SARAH:~/MPI_Lab5_Golenishchev_KE220/build/MPI_Lab-Debug$
```

Рисунок 2. Результат выполнения программы передачи сообщений от «рабочих»
«мастеру»

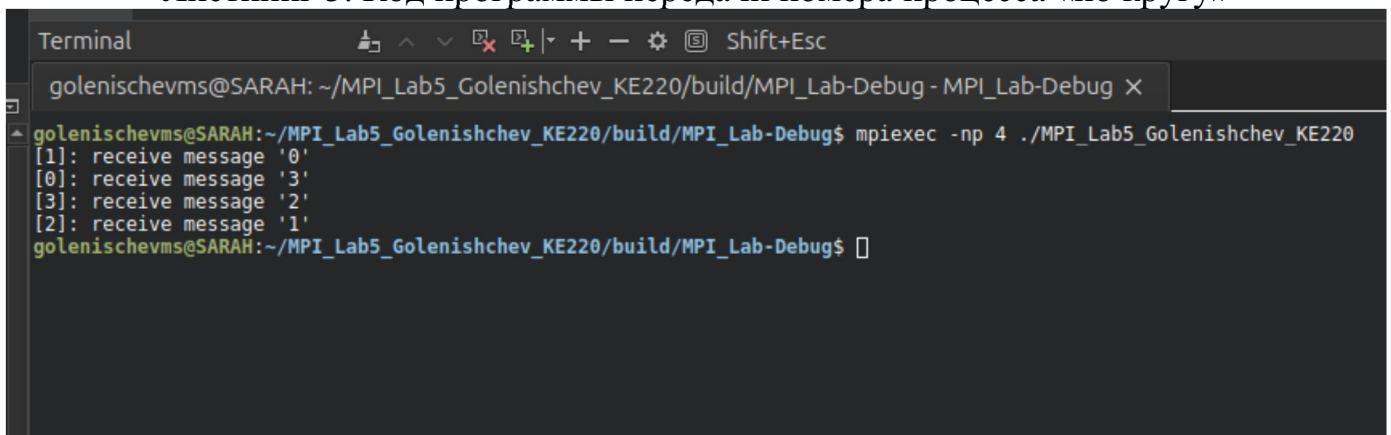
Задание 15. Коммуникации «точка-точка»: схема «сдвиг по кольцу»

Разработали программу обмена сообщениями между процессами по кругу, листинг 3. Представлен результат ее работы, рисунок 3.

```
#include <mpi.h>
#include <stdio.h>
// Golenishchev Artem, KE-220 Task 15
int main(int argc, char** argv) {
    int rank, size, buf, recv_buf;
    MPI_Request requests[2];
    MPI_Status statuses[2];

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); // Получение номера текущего процесса
    MPI_Comm_size(MPI_COMM_WORLD, &size); // Получение общего числа процессов
    // Определяем номера соседних процессов в кольце
    int next = (rank + 1) % size; // Следующий процесс
    int prev = (rank - 1 + size) % size; // Предыдущий процесс (учитываем кольцо)
    buf = rank; // Сообщение, отправляемое текущим процессом
    recv_buf = -1; // Инициализация буфера для получения сообщения
    // Неблокирующая отправка сообщения следующему процессу
    MPI_Isend(&buf, 1, MPI_INT, next, 0, MPI_COMM_WORLD, &requests[0]);
    // Неблокирующий прием сообщения от предыдущего процесса
    MPI_Irecv(&recv_buf, 1, MPI_INT, prev, 0, MPI_COMM_WORLD, &requests[1]);
    // Ожидаем завершения всех операций
    MPI_Waitall(2, requests, statuses);
    printf("[%d]: receive message '%d'\n", rank, recv_buf);
    MPI_Finalize();
    return 0;
}
```

Листинг 3. Код программы передачи номера процесса «по кругу»



```
Terminal
golenishevms@SARAH: ~/MPI_Lab5_Golenishchev_KE220/build/MPI_Lab-Debug - MPI_Lab-Debug X
golenishevms@SARAH:~/MPI_Lab5_Golenishchev_KE220/build/MPI_Lab-Debug$ mpiexec -np 4 ./MPI_Lab5_Golenishchev_KE220
[1]: receive message '0'
[0]: receive message '3'
[3]: receive message '2'
[2]: receive message '1'
golenishevms@SARAH:~/MPI_Lab5_Golenishchev_KE220/build/MPI_Lab-Debug$
```

Рисунок 3. Результат выполнения программы передачи сообщений между процессами в круговом порядке

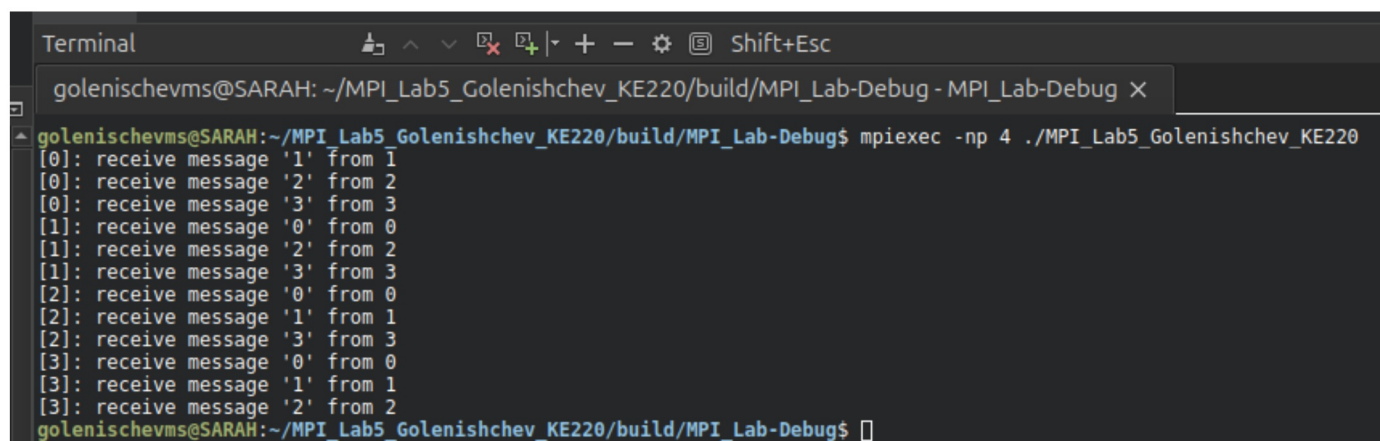
Задание 16. Коммуникации «точка-точка»: схема «каждый каждому»

Разработали программу обмена сообщениями между процессами от всех каждому, листнинг 4. Представлен результат ее работы, рисунок 4.

```
#include <mpi.h>
#include <stdio.h>
// Golenishchev Artem, KE-220 Task 16
int main(int argc, char** argv) {
    int rank, size, buf, recv_buf;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); // Получение номера текущего процесса
    MPI_Comm_size(MPI_COMM_WORLD, &size); // Получение общего числа процессов
    buf = rank; // Сообщение, которое отправляет текущий процесс
    // Передача сообщений от текущего процесса всем другим процессам
    for (int i = 0; i < size; i++) {
        if (i != rank) { // Исключаем отправку самому себе
            MPI_Send(&buf, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
        }
    }
    // Прием сообщений от всех других процессов
    for (int i = 0; i < size; i++) {
        if (i != rank) { // Исключаем прием от самого себя
            MPI_Recv(&recv_buf, 1, MPI_INT, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            printf("[%d]: receive message '%d' from %d\n", rank, recv_buf, i);
        }
    }
    MPI_Finalize();
    return 0;
}
```

Листнинг 4. Код программы обмена сообщениями



```
Terminal
golenischchevms@SARAH: ~/MPI_Lab5_Golenishchev_KE220/build/MPI_Lab-Debug - MPI_Lab-Debug x
golenischchevms@SARAH:~/MPI_Lab5_Golenishchev_KE220/build/MPI_Lab-Debug$ mpiexec -np 4 ./MPI_Lab5_Golenishchev_KE220
[0]: receive message '1' from 1
[0]: receive message '2' from 2
[0]: receive message '3' from 3
[1]: receive message '0' from 0
[1]: receive message '2' from 2
[1]: receive message '3' from 3
[2]: receive message '0' from 0
[2]: receive message '1' from 1
[2]: receive message '3' from 3
[3]: receive message '0' from 0
[3]: receive message '1' from 1
[3]: receive message '2' from 2
golenischchevms@SARAH:~/MPI_Lab5_Golenishchev_KE220/build/MPI_Lab-Debug$
```

Рисунок 4. Результат выполнения программы обмена номерами между процессами

Ответы на вопросы:

1. Рассказать общую схему межпроцессного обмена (перечислить участников, указать порядок действий).

Участниками обмена являются процессы, входящие в группу коммуникации (например, `MPI_COMM_WORLD`). Каждый процесс передает данные другим процессам (отправители) и принимает данные от них (получатели). Сначала инициализируется MPI с помощью `MPI_Init`, определяется номер текущего процесса (`rank`) и общее количество процессов (`size`). Затем происходит обмен: процессы используют функции отправки (`MPI_Send`) и приема (`MPI_Recv`) для передачи данных. Завершается работа вызовом `MPI_Finalize`.

2. Данные какого типа передают все функции MPI? Привести общий формат функции передачи сообщений. Какие объекты идентифицируются этими параметрами?

Тип данных и формат функции MPI: Функции MPI передают данные различных типов, определяемых параметром `MPI_Datatype` (например, `MPI_INT`, `MPI_FLOAT`). Формат функции передачи сообщений: `int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm);`, где параметры описывают буфер с данными, количество элементов, тип данных, процесс-получатель, идентификатор сообщения и группу процессов.

3. Рассказать про реализованные в MPI классификации механизмов передачи сообщений.

MPI поддерживает блокирующую (например, `MPI_Send`, `MPI_Recv`) и неблокирующую (`MPI_Isend`, `MPI_Irecv`) передачи, а также синхронную (`MPI_Ssend`) и буферизованную (`MPI_Bsend`) передачи. Также реализованы групповые коммуникации, включая широковещательные (`MPI_Bcast`), редуционные (`MPI_Reduce`) и обмены между всеми процессами (`MPI_Alltoall`).

4. Для каждой задачи пояснить, почему для этого использован именно этот вид функций, получен ли прирост эффективности за счет их использования?

В задаче «Сдвиг по кольцу» и схеме «каждый каждому» использованы блокирующие функции (`MPI_Send` и `MPI_Recv`) для простоты реализации и надежного упорядоченного обмена. Они гарантируют завершение передачи данных перед продолжением работы, что упрощает синхронизацию. Прироста эффективности за счет их использования нет, но они обеспечивают надежную логику.

5. Как избежать синхронизации в порядке приема сообщений без смены функции? Как в этом случае определить отправителя сообщения?

Синхронизации можно избежать, используя параметр `MPI_ANY_SOURCE` в функции `MPI_Recv`, чтобы принимать сообщения от любого процесса. Отправителя сообщения можно определить через поле `MPI_SOURCE` структуры `MPI_Status`, переданной в функцию. Это позволяет принимать сообщения в произвольном порядке, избегая привязки к конкретным отправителям.

Выводы:

Изучили основные механизмы межпроцессного взаимодействия в MPI, включая блокирующие и неблокирующие функции передачи сообщений типа «точка-точка». Рассмотрены принципы организации схем обмена данными между процессами, такие как «сдвиг по кольцу» и «каждый каждому». Изучены функции передачи (MPI_Send, MPI_Recv) и их параметры, а также подходы к идентификации отправителей и получателей сообщений. Были исследованы способы синхронизации операций и устранения блокировок. Работа позволила закрепить навыки организации эффективного обмена данными в параллельных системах и подчеркнула важность выбора подходящих функций для оптимизации процессов коммуникации.