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.

***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

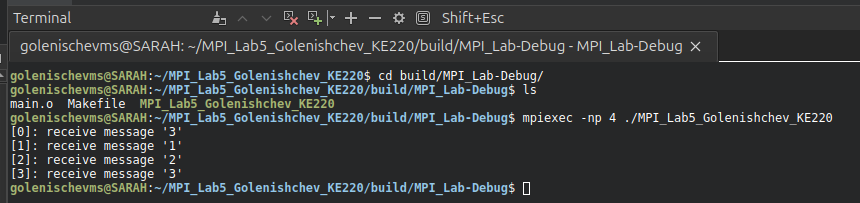


Рисунок 1. Результат работы программы обмена сообщений между процессами

***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»

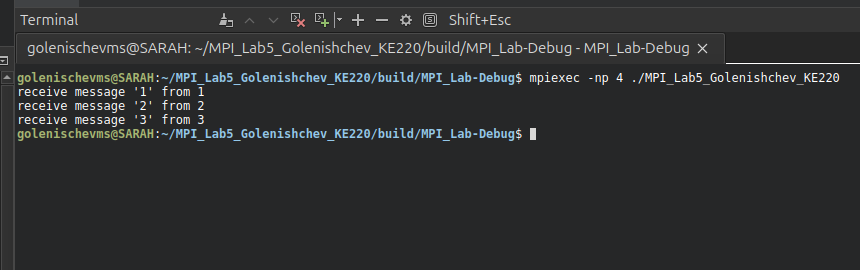


Рисунок 2. Результат выполнения программы передачи сообщений от «рабочих» «мастеру»

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

Разработали программу обмена сообщениями между процессами по кругу, листнинг 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;

}

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

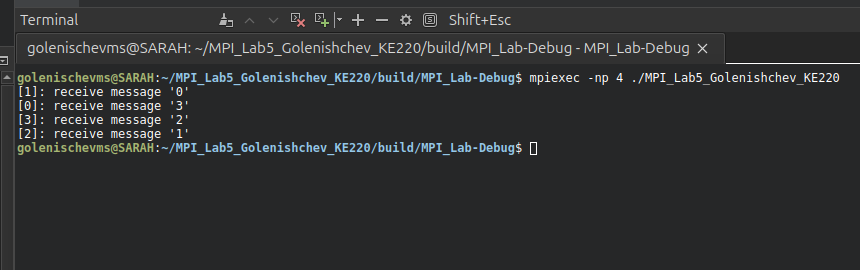


Рисунок 3. Результат выполнения программы передачи сообщений   
между процессами в круговом порядке

***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.

Листнинг 4. Код программы обмена сообщениями

#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);

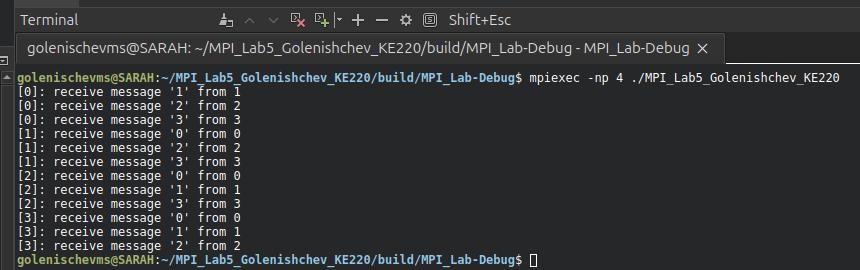
}

}

MPI\_Finalize();

return 0;

}

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

1. 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.

1. 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), редукционные (MPI\_Reduce) и обмены между всеми процессами (MPI\_Alltoall).

1. 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.

1. 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.