**Лабораторная работа №1**

**Передача и прием сообщений в MPI**

**Отчет и код выполнил**

**студент группы ПИН-34**

**Устян Давид**

Цель: изучить основные принципы приема и передачи сообщений в технологии MPI на примере использования в рамках языка С++.

Под параллельной программой в рамках MPI понимается множество одновременно выполняемых процессов. Процессы могут выполняться как на разных процессорах, так и на одном. Каждый процесс параллельной программы порождается на основе копии одного и того же программного кода (модель SPMD). Все процессы программы последовательно перенумерованы от 0 до p-1, где p есть общее количество процессов. Номер процесса именуется рангом процесса

**OpenMP** -это способ программирования на устройствах с **общей памятью**.

**MPI** -это способ программирования на устройствах с **распределенной памятью**.

**Задание (вариант 2):**

Реализуйте процесс-«счётчик», (который запускается со значением 0) и

1) если получена -1, то он выводит в текущее значение и заканчивает работу;

2) если получено любое другое сообщение, то значение увеличивается на 1 и выводится сообщение об этом.

Я не уверен, что правильно понял задание, но так как главной целью работы является прием и отправка сообщений, то я сконцентрировался на этом. Интерпретация задания вышла следующей:

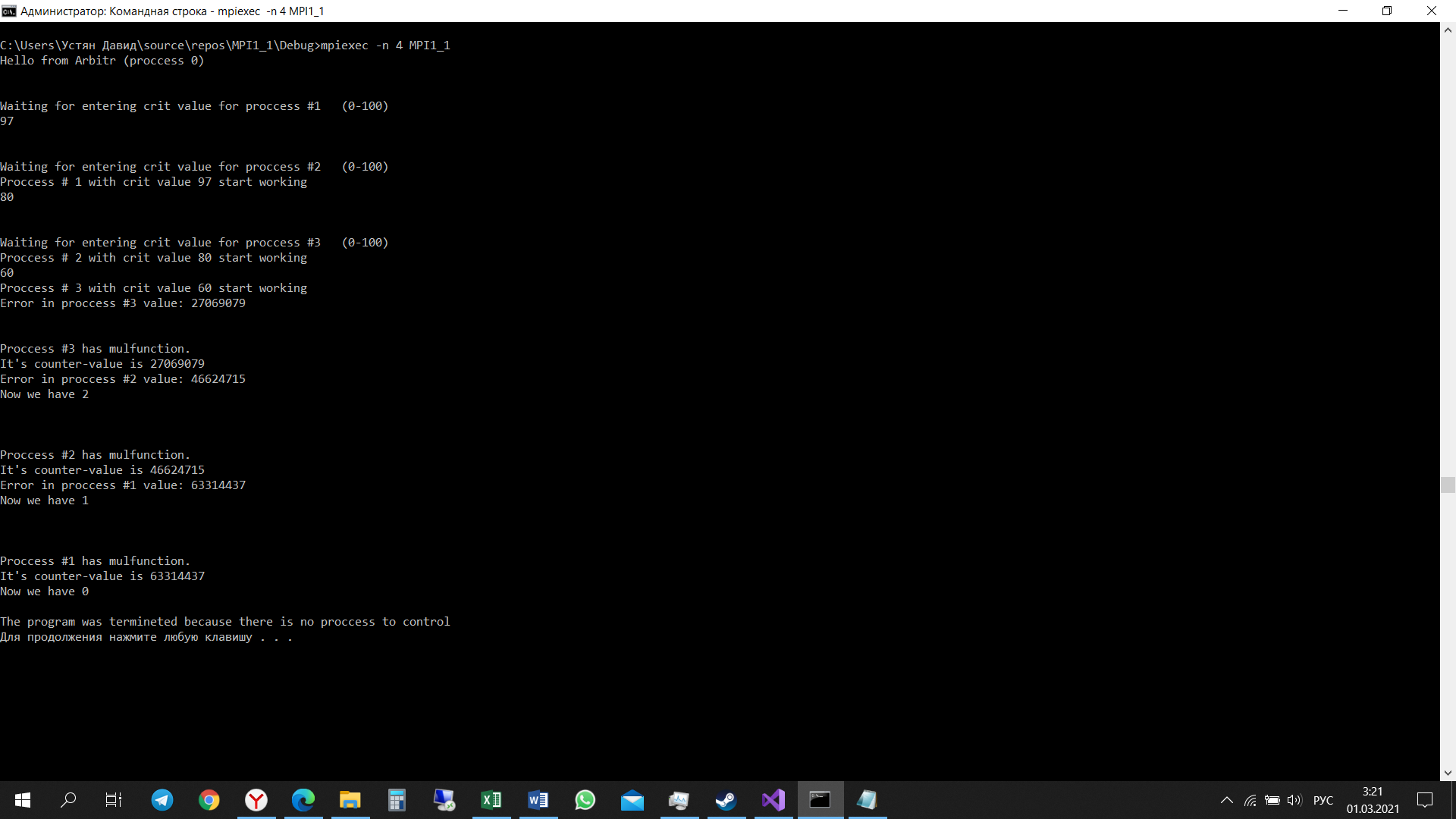
Программа симулирует подсчет выполненных работ до первой поломки, то есть **если получено любое другое сообщение, то значение увеличивается на 1 и выводится сообщение об этом.**

В случае поломки (**если получена -1, то заканчивает работу**) ненулевой процесс отправляет отчет и завершает работу

Процесс 0 анализирует выполнение работы другими процессами и завершает работу, если он остался один.

Для получения «значений» была сделана отдельная функция-рандомайзер.

Результат запуска для 4-х процессов (включая «нулевой»)



Как видно по консоли, при получении сообщения с критическим значением от Арбитра процессы тут же начинали работу, чем порой затрудняли ввод из-за общего экрана ввода-вывода. Кроме этого, процессы также перебивали друг друга. Это можно было исправить, если поручить вывод только нулевому процессу (делать это мне очень не хочется, поэтому хотелось бы сдать так)

При выпадении «ошибки» подчиненный процесс сообщает свой ранг арбитру, тот запрашивает значение счетчика и только после получения выводит его на экран.

Код самой программы:

#include <iostream>

#include "mpi.h"

#include "time.h"

#include "stdlib.h"

#include "string"

using namespace std;

// Программа симулирует подсчет выполненных работ до первой поломки.

// За симуляцию работы отвечает функция work(int crit), где crit - критическое значение от 0 до 100

// В случае поломки функция возвращает -1, после чего сломавшийся процесс отправляет отчет и завершает работу

// Процесс 0 анализирует выполнение работы другими процессами и завершает работу, если он остался один.

int work(int crit)

{

srand(time(NULL));

int value = rand() % 100;

if (value < crit)

return value;

return -1;

}

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

{

int buffer, numtasks, rank;

int done\_work, work\_value;

int crit\_value = 0;

MPI\_Status Status;

// <программный код без использования MPI функций>

MPI\_Init(&argc, &argv);

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

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

// <программный код с использованием MPI функций>

if (rank == 0)

{

cout << "Hello from Arbitr (proccess 0)" << endl;

for (int i = 1; i < numtasks; i++)

{

do {

cout << "\n\nWaiting for entering crit value for proccess #"<< i << " (0-100) " << endl;

cin >> crit\_value;

} while (crit\_value < 0 || crit\_value>100);

MPI\_Send(&crit\_value, 1, MPI\_INT, i, 10, MPI\_COMM\_WORLD);

}

while (numtasks > 1)

{

buffer = 0;

MPI\_Recv(&buffer, 1, MPI\_INT, MPI\_ANY\_SOURCE,

MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &Status);

cout << "\n\nProccess #" << buffer << " has mulfunction." << endl;

work\_value = 0;

MPI\_Send(&work\_value, 1, MPI\_INT, buffer, 1, MPI\_COMM\_WORLD);

MPI\_Recv(&work\_value, 1, MPI\_INT, buffer,

1, MPI\_COMM\_WORLD, &Status);

cout << "It's counter-value is " << work\_value << endl;

numtasks--;

cout << "Now we have " << numtasks - 1 << "\n\n";

}

cout << "The program was termineted because there is no proccess to control" << endl;

system("pause");

}

else

{

MPI\_Recv(&crit\_value, 1, MPI\_INT, 0,

10, MPI\_COMM\_WORLD, &Status);

cout << "Proccess # " << rank << " with crit value "<< crit\_value << " start working" << endl;

work\_value = 0;

done\_work = -1;

do {

done\_work++;

work\_value = work(crit\_value);

} while (work\_value != -1);

cout << "Error in proccess #" << rank << " value: "<< done\_work <<endl;

buffer = rank;

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

MPI\_Recv(&buffer, 1, MPI\_INT, MPI\_ANY\_SOURCE,

1, MPI\_COMM\_WORLD, &Status);

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

}

MPI\_Finalize();

// <программный код без использования MPI функций>

return 0;

}

Были попытки передать сразу несколько значений в буфер, являющий собой некоторый массив. Сделать это «по-нормальному» не вышло. Кроме того, не удалось найти пример подобного выполнения. Предполагаю, что при записи функция MPI\_Send заполняет указанное количество ячеек массива, а функция MPI\_Send - считывает. В будущем собираюсь разобраться с данным вопросом.

Контрольные вопросы

1. В чем состоят основы технологии MPI?

Чтобы программа начала на нем выполняться, ее необходимо **скопировать** на каждый узел, **запустить** и **установить связь** между процессами. Эту работу берет на себя утилита mpirun (под Linux) или mpiexec (под Windows) https://pro-prof.com/archives/4386#mpi\_intro

1. В чем состоят основные преимущества и недостатки технологии MPI?

Основным преимуществом считаю тот факт, что MPI работает с разделенной памятью, а значит отсутствует такое явление, как гонка. Под недостатком понимаю время, которое требуется для отправки сообщений между кластерами, однако при слабой связности кластеров сообщений между ними тоже будет немного.

1. Что понимается под параллельной программой в рамках технологии MPI?

Под параллельной программой понимается разделение работы между «подчиненными» процессами под руководством процесса ранга 0. Подчиненные процессы параллельно выполняют вычисления, после чего отправляют результат нулевому. (принцип SPMD)

1. Как происходит инициализация и завершение MPI программ?

Инициализацию производит функция MPI\_Init(&argc, &argv);

Завершение проводит функция MPI\_Finalize();

5. Как происходит передача и прием сообщений MPI программе?Отправка и прием сообщения происходит посредством функций MPI\_Send и MPI\_Recv соответственно.