Saint Petersburg National Research University of Information Technologies, Mechanics and Optics (ITMO University)

REPORT

about laboratory works

Assignment 3
Assignment 4.
Assignment 5.

Student Pogrebnoy D.A. j4132c

Saint-Petersburg, 2021

ASSIGNMENT 3.

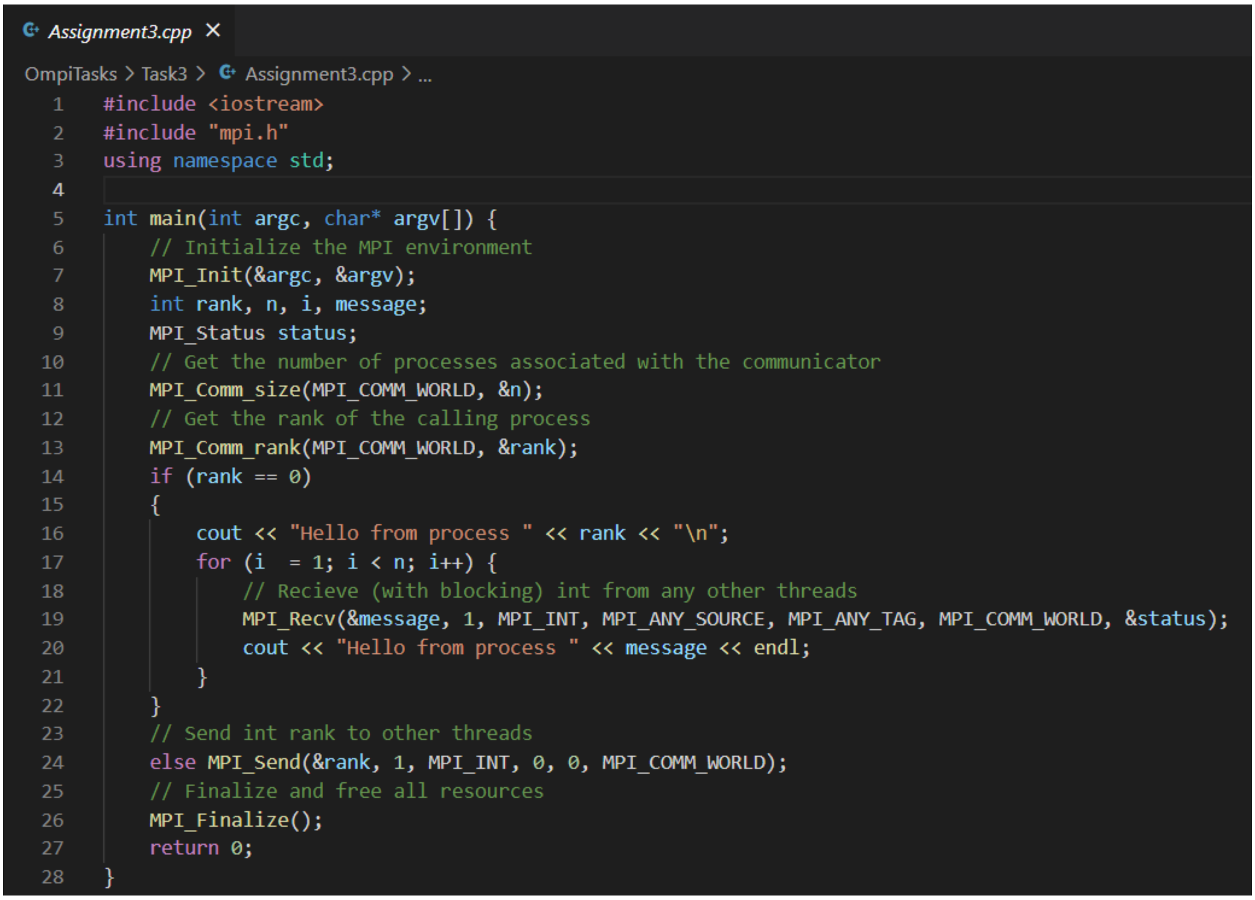
Task

Compile and run Assignment3.c program. Explain in detail how it works.

Implementation

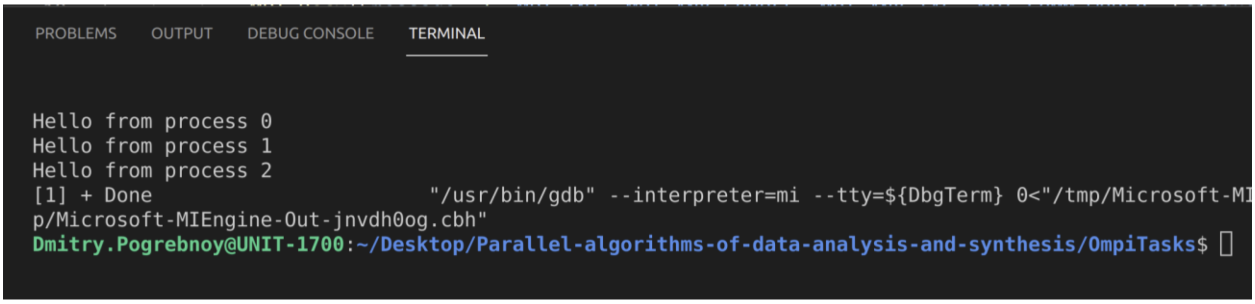
Source code and data gathered are available on https://github.com/DmitryPogrebnoy/Parallel-algorithms-of-data-analysis-and-synthesis/blob/master/OmpiTasks/Task3/Assignment3.cpp

The description of the code is described in the comments. The main process waits for messages to be received, and the other processes send messages to it with their thread number.



```cpp
#include <iostream>
#include "mpi.h"
using namespace std;

int main(int argc, char* argv[]) {
    // Initialize the MPI environment
    MPI_Init(&argc, &argv);
    int rank, n, i, message;
    MPI_Status status;
    // Get the number of processes associated with the communicator
    MPI_Comm_size(MPI_COMM_WORLD, &n);
    // Get the rank of the calling process
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0)
    {
        cout << "Hello from process " << rank << "\n";
        for (i  = 1; i < n; i++) {
            // Recieve (with blocking) int from any other threads
            MPI_Recv(&message, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
            cout << "Hello from process " << message << endl;
        }
    }
    // Send int rank to other threads
    else MPI_Send(&rank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    // Finalize and free all resources
    MPI_Finalize();
    return 0;
}
```

Output example with 3 processes:



```
Hello from process 0
Hello from process 1
Hello from process 2
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MI
p/Microsoft-MIEngine-Out-jnvdh0og.cbh"
Dmitry.Pogrebnoy@UNIT-1700:~/Desktop/Parallel-algorithms-of-data-analysis-and-synthesis/OmpiTasks$ 
```

ASSIGMENTS 4.

Task

Convert the code Assignment4.c to match your individual version of the assignment.

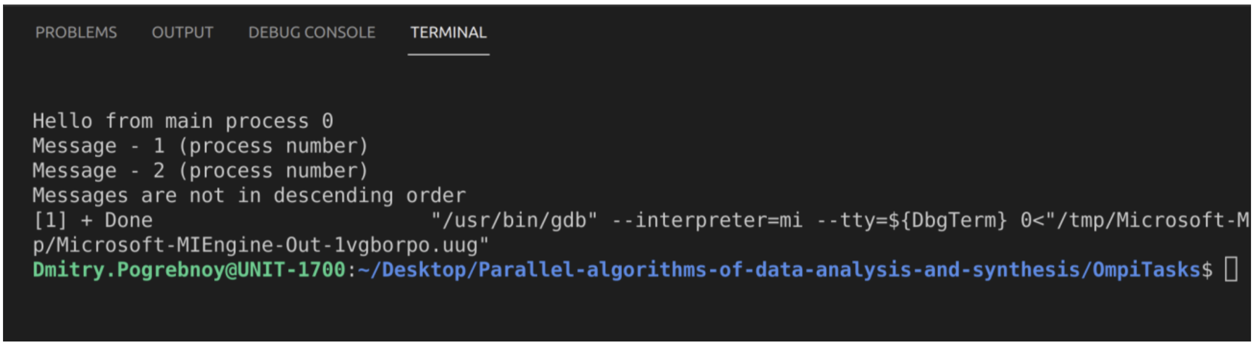
Option #21. The root process accepts messages from child processes and determines whether the sequence is strictly descending.

Implementation

Source code and data gathered are available on https://github.com/DmitryPogrebnoy/Parallel-algorithms-of-data-analysis-and-synthesis/blob/master/OmpiTasks/Task4/Assignment4.cpp

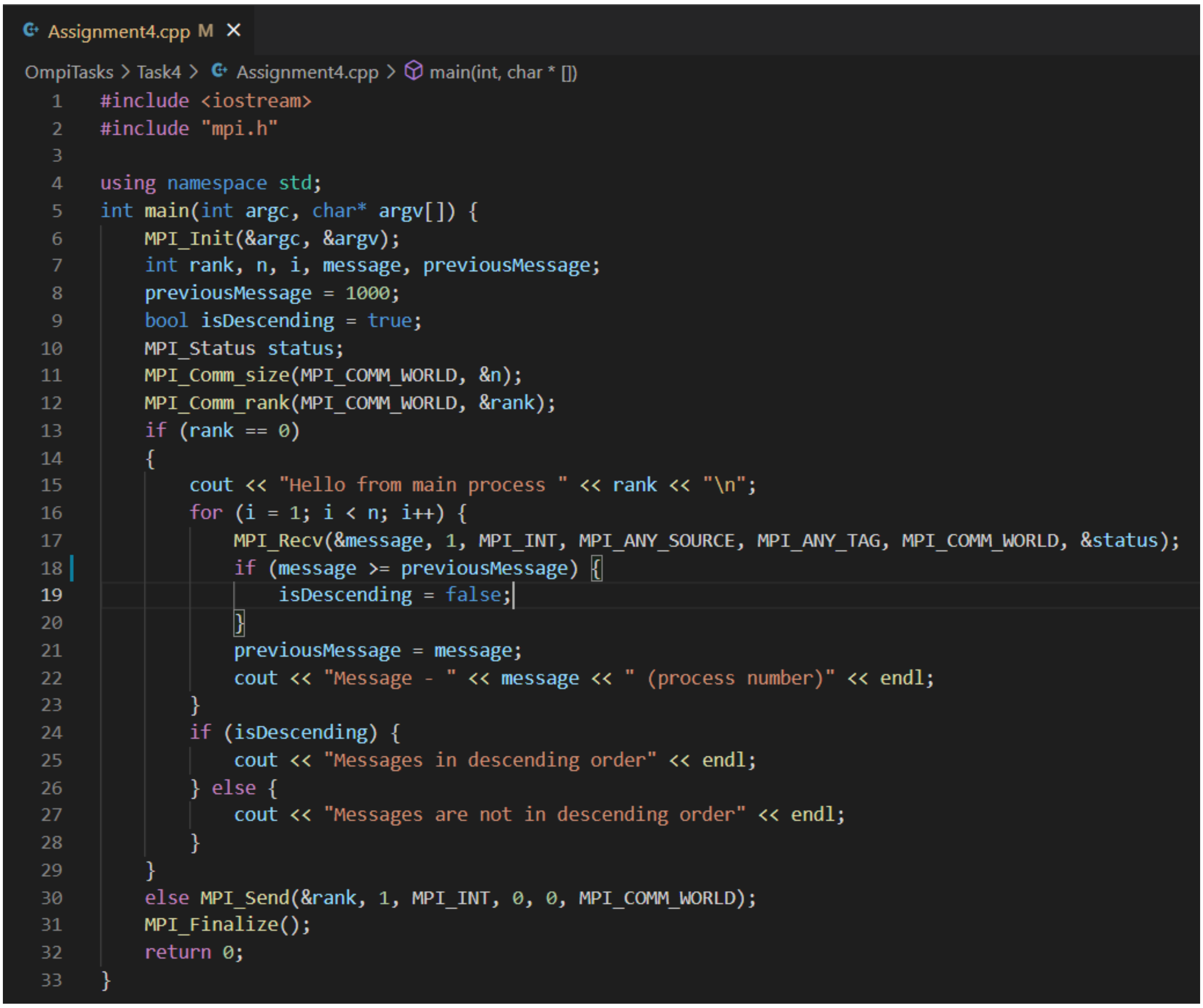
The main process saves the previous message and compares it with the new one, if the order is broken, then the corresponding flag is set. And at the end, the main process outputs the corresponding message.

Output example with 3 processes:



```
Hello from main process 0
Message - 1 (process number)
Message - 2 (process number)
Messages are not in descending order
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-M
p/Microsoft-MIEngine-Out-1vgborpo.uug"
Dmitry.Pogrebnoy@UNIT-1700:~/Desktop/Parallel-algorithms-of-data-analysis-and-synthesis/OmpiTasks$ 
```

The code looks like this:



```
#include <iostream>
#include "mpi.h"

using namespace std;
int main(int argc, char* argv[]) {
    MPI_Init(&argc, &argv);
    int rank, n, i, message, previousMessage;
    previousMessage = 1000;
    bool isDescending = true;
    MPI_Status status;
    MPI_Comm_size(MPI_COMM_WORLD, &n);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0)
    {
        cout << "Hello from main process " << rank << "\n";
        for (i = 1; i < n; i++) {
            MPI_Recv(&message, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
            if (message >= previousMessage) {
                isDescending = false;
            }
            previousMessage = message;
            cout << "Message - " << message << " (process number)" << endl;
        }
        if (isDescending) {
            cout << "Messages in descending order" << endl;
        } else {
            cout << "Messages are not in descending order" << endl;
        }
    }
    else MPI_Send(&rank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    MPI_Finalize();
    return 0;
}
```

ASSIGNMENT 5.

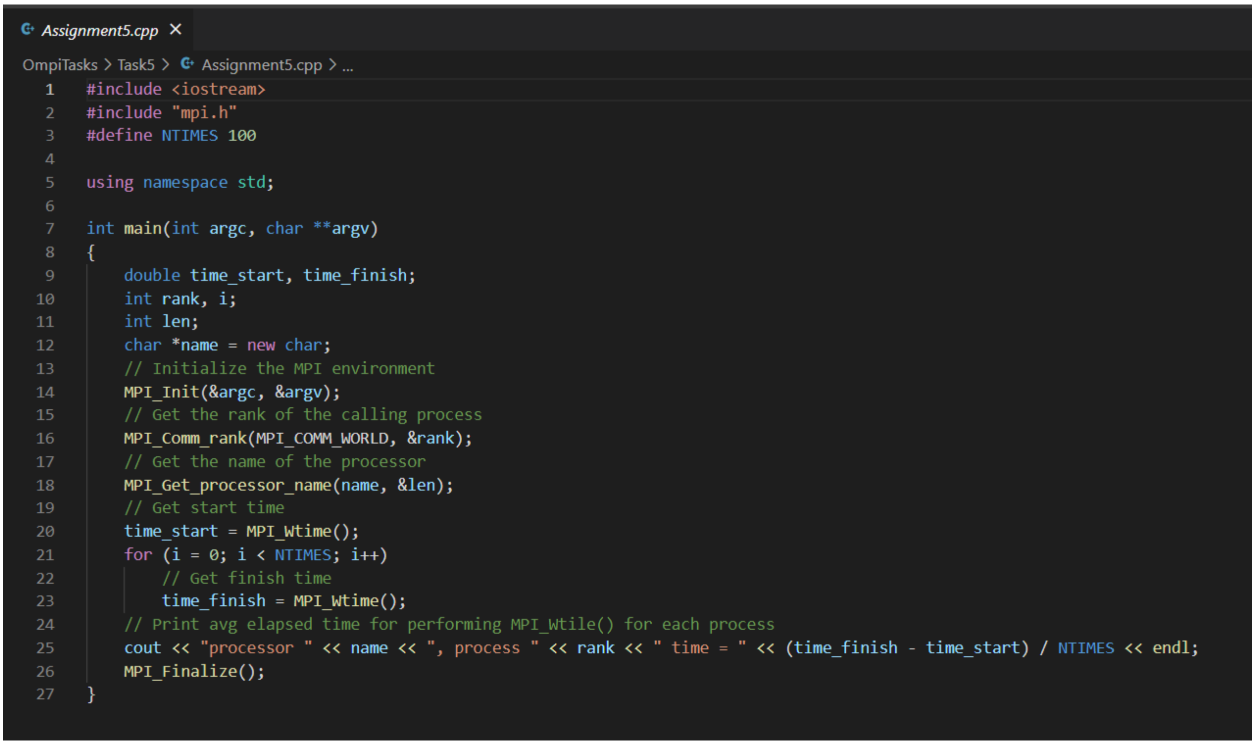
Task

Compile and run Assignment5.c program. Explain in detail how it works. Determine the execution time of the program from the previous task.

Implementation

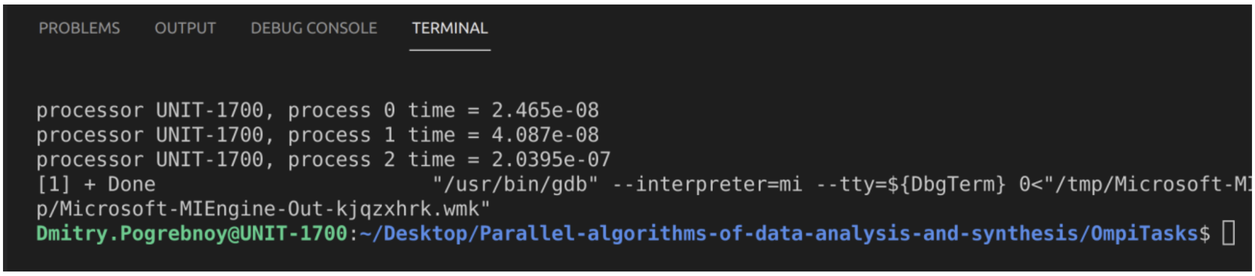
Source code and data gathered are available on https://github.com/DmitryPogrebnoy/Parallel-algorithms-of-data-analysis-and-synthesis/blob/master/OmpiTasks/Task5 .

The description of the code is described in the comments. In each process, the time measurement is called 100 times using MPI_Wtime() and the average time value is output.



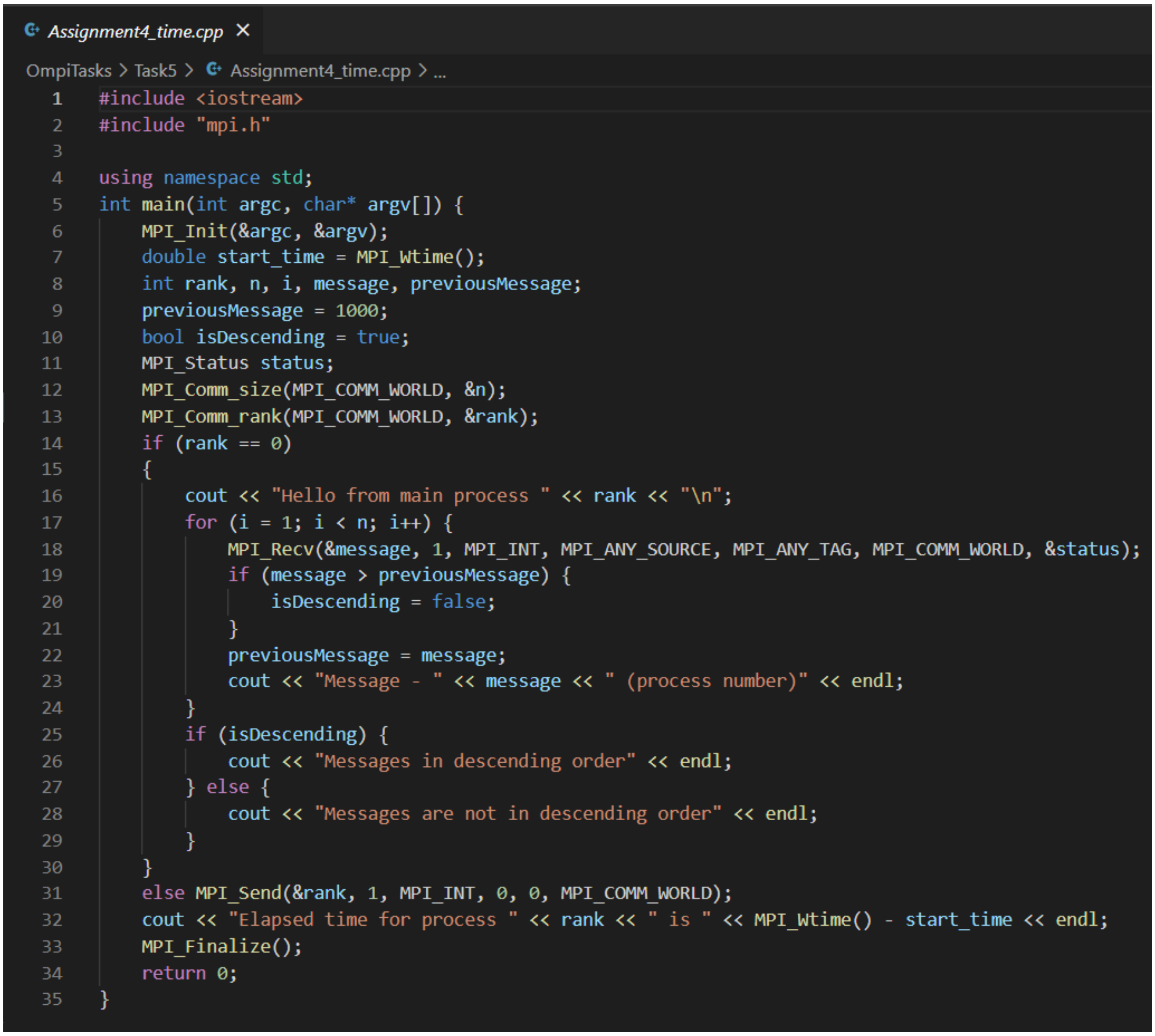
```cpp
#include <iostream>
#include "mpi.h"
#define NTIMES 100

using namespace std;

int main(int argc, char **argv)
{
    double time_start, time_finish;
    int rank, i;
    int len;
    char *name = new char;
    // Initialize the MPI environment
    MPI_Init(&argc, &argv);
    // Get the rank of the calling process
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    // Get the name of the processor
    MPI_Get_processor_name(name, &len);
    // Get start time
    time_start = MPI_Wtime();
    for (i = 0; i < NTIMES; i++)
        // Get finish time
        time_finish = MPI_Wtime();
    // Print avg elapsed time for performing MPI_Wtile() for each process
    cout << "processor " << name << ", process " << rank << " time = " << (time_finish - time_start) / NTIMES << endl;
    MPI_Finalize();
}
```

Output example with 3 processes:



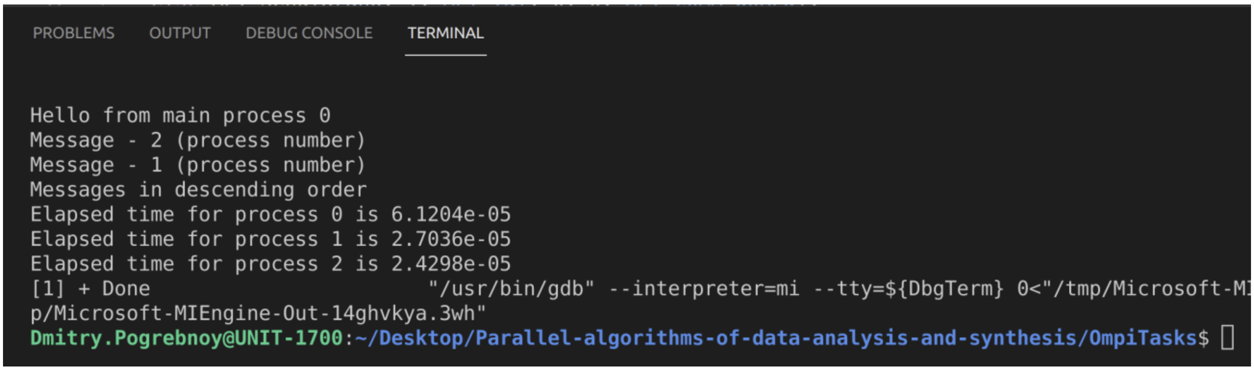
```
processor UNIT-1700, process 0 time = 2.465e-08
processor UNIT-1700, process 1 time = 4.087e-08
processor UNIT-1700, process 2 time = 2.0395e-07
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MI
p/Microsoft-MIEngine-Out-kjqzxhrk.wmk"
Dmitry.Pogrebnoy@UNIT-1700:~/Desktop/Parallel-algorithms-of-data-analysis-and-synthesis/OmpiTasks$ 
```

Assignment4 with time measurement looks like this:



```cpp
#include <iostream>
#include "mpi.h"

using namespace std;
int main(int argc, char* argv[]) {
    MPI_Init(&argc, &argv);
    double start_time = MPI_Wtime();
    int rank, n, i, message, previousMessage;
    previousMessage = 1000;
    bool isDescending = true;
    MPI_Status status;
    MPI_Comm_size(MPI_COMM_WORLD, &n);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0)
    {
        cout << "Hello from main process " << rank << "\n";
        for (i = 1; i < n; i++) {
            MPI_Recv(&message, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
            if (message > previousMessage) {
                isDescending = false;
            }
            previousMessage = message;
            cout << "Message - " << message << " (process number)" << endl;
        }
        if (isDescending) {
            cout << "Messages in descending order" << endl;
        } else {
            cout << "Messages are not in descending order" << endl;
        }
    }
    else MPI_Send(&rank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
    cout << "Elapsed time for process " << rank << " is " << MPI_Wtime() - start_time << endl;
    MPI_Finalize();
    return 0;
}
```

Output example with 3 processes:



```
Hello from main process 0
Message - 2 (process number)
Message - 1 (process number)
Messages in descending order
Elapsed time for process 0 is 6.1204e-05
Elapsed time for process 1 is 2.7036e-05
Elapsed time for process 2 is 2.4298e-05
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MI
p/Microsoft-MIEngine-Out-14ghvkya.3wh"
Dmitry.Pogrebnoy@UNIT-1700:~/Desktop/Parallel-algorithms-of-data-analysis-and-synthesis/OmpiTasks$ 
```