МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра вычислительной техники

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Параллельные вычисления»
Тема: «Запуск параллельной программы»

Студентка гр. 1307 _____ Голубев М.А.

Преподаватель _____ Калмыков М.А.

Санкт-Петербург
2025

Цель работы.

Освоить процесс запуска программы на C++ с применением библиотеки MPICH2. Научиться получать сведения о количестве запущенных процессов и номере отдельного процесса.

Задание 1. Создать и запустить программу на 2-х процессах с применением функций int MPI_Init(int* argc, char* argv) и int MPI_Finalize(void).**

Текст программы lab1_1.c

```c
#include <stdio.h>
#include "mpi.h"

int main(int argc, char **argv)
{
   MPI_Init(&argc, &argv);
   MPI_Finalize();
   printf("Успешно\n");
   return 0;
}
```

Компиляция и запуск программы на 2-х процессах.

```
21   # Указываем команду для запуска программы через mpiexec
22   CMD ["sh", "-c", "mpicc $F_NAME -o main && mpiexec -n $N ./main"]
23
```

```
ПРОБЛЕМЫ 7   ВЫХОДНЫЕ ДАННЫЕ   КОНСОЛЬ ОТЛАДКИ   ТЕРМИНАЛ   ПОРТЫ   КОММЕНТ
mishhgun@Air-Mihail lab1 % docker run --rm mpich-1-1
Успешно
Успешно
```

Задание 2. Создать и запустить программу на 3-х процессах с применением функций:

1) **int MPI_Init(int* argc, char*** argv);**
2) **int MPI_Finalize(void);**
3) **int MPI_Comm_size(MPI_Comm comm, int* size)**
4) **int MPI_Comm_rank(MPI_Comm comm, int* rank)**

Программа должна выводить на экран номер процесса и какой-либо идентификатор процесса.

Текст программы lab1_2.c

```c
#include <stdio.h>
#include "mpi.h"
#include <sys/types.h>
#include <unistd.h>

int main(int argc, char **argv)
{
   int rank, size;

   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &size);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   printf("Номер процесса - %i, PID - %i \n", rank, getpid());

   MPI_Finalize();

   return 0;
}
```

Компиляция и запуск программы на 3-х процессах.

```
14
15    CMD ["sh", "-c", "mpicc $F_NAME -o main && mpiexec -n $N ./main"]
16
```

```
ПРОБЛЕМЫ 9   ВЫХОДНЫЕ ДАННЫЕ   КОНСОЛЬ ОТЛАДКИ   ТЕРМИНАЛ   ПОРТЫ   КОММЕ
mishhgun@Air-Mihail lab1 % docker run --rm mpich-1-2
номер процесса - 0, PID - 29
номер процесса - 1, PID - 30
номер процесса - 2, PID - 31
mishhgun@Air-Mihail lab1 %
```

Задание 3. Создать и запустить программу на n-х процессах печати таблицы умножения.

Текст программы lab1_3.c

```c
#include <stdio.h>
#include "mpi.h"
```

```c
int main(int argc, char **argv)
{
   int rank, size;

   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &size);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   for (int j = rank + 1; j <= 10; j = j + size)
   {
       for (int i = 1; i <= 10; ++i)
       {
           printf("|%d*%d=%d", i, j, j * i);
       }
       printf("\n");
   }
   MPI_Finalize();

   return 0;
}
```

Запуск программы на 4-х процессах.

```
mishhgun@MacBook-Air-Mihail lab1 % docker run mpich-1-3
|1*1=1|2*1=2|3*1=3|4*1=4|5*1=5|6*1=6|7*1=7|8*1=8|9*1=9|10*1=10
|1*5=5|2*5=10|3*5=15|4*5=20|5*5=25|6*5=30|7*5=35|8*5=40|9*5=45|10*5=50
|1*9=9|2*9=18|3*9=27|4*9=36|5*9=45|6*9=54|7*9=63|8*9=72|9*9=81|10*9=90
|1*3=3|2*3=6|3*3=9|4*3=12|5*3=15|6*3=18|7*3=21|8*3=24|9*3=27|10*3=30
|1*7=7|2*7=14|3*7=21|4*7=28|5*7=35|6*7=42|7*7=49|8*7=56|9*7=63|10*7=70
|1*2=2|2*2=4|3*2=6|4*2=8|5*2=10|6*2=12|7*2=14|8*2=16|9*2=18|10*2=20
|1*6=6|2*6=12|3*6=18|4*6=24|5*6=30|6*6=36|7*6=42|8*6=48|9*6=54|10*6=60
|1*10=10|2*10=20|3*10=30|4*10=40|5*10=50|6*10=60|7*10=70|8*10=80|9*10=90|1*4=4|2*4=8|3*4=12|4*4=16|5*4=20
|6*4=24|7*4=28|8*4=32|10*10=100
|9*4=36|10*4=40
|1*8=8|2*8=16|3*8=24|4*8=32|5*8=40|6*8=48|7*8=56|8*8=64|9*8=72|10*8=80
```

Выводы.

В ходе выполнения лабораторной работы были изучены основные концепции параллельного программирования с применением MPI. Полученные умения по запуску процессов, распределению задач и контролю над ними представляют собой основу для создания сложных распределённых

приложений, которые требуют оптимизации вычислений с помощью параллелизма.