# Паралелно и дистрибуирано процесирање

*Лабораториска вежба 2*

*Петар Атанасовски - 216052*

***Програмите се напишани во програмскиот јазик c***

1. **Напишете програма во којашто ќе изготвите низа од 3 карактери. Во рамките на процесот со ранк 0 нека се промени низата со други произволни податоци, различни од првичните. По промената, таа низа да се испрати до сите процеси, коишто откако ќе ја примат, ќе ја испечатат низата.**

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

int main(int argc, char **argv)
{
   MPI_Init(NULL, NULL);

   int world_size;
   MPI_Comm_size(MPI_COMM_WORLD, &world_size);

   int rank;
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);

   const int ROOT_ID = 0;
   char initial_array[3] = {'a', 'b', 'c'};

   if (rank == ROOT_ID)
   {
      for (int i = 0; i < 3; i++)
      {
         initial_array[i] = initial_array[i] + 2;
      }
   }

   MPI_Bcast(&initial_array, 3, MPI_CHAR, ROOT_ID, MPI_COMM_WORLD);

   printf("Process %d received: %c%c%c\n", rank, initial_array[0], initial_array[1],
initial_array[2]);

   MPI_Finalize();
   return 0;
}
```

2. **Модифицирајте го примерот за gather и scatter така што наместо максимум ќе пронајдете просек на низата. Првиот процес ја разделува на еднакви делчиња на останатите процеси. Тие наоѓаат локален просек, а со помош на локалните**

**проседи, првиот процес пресметува просек на целата низа откако ќе ги прими локалните проседи.**

```
#include <stdio.h>
#include <mpi.h>
#include <stdlib.h>
#define BR_ELEMENTI 100

double najdi_avg(double *niza, int br_elem)
{
   int sum = 0;
   for (int i = 0; i < br_elem; i++)
   {
      sum += niza[i];
   }
   return (sum * 1.0) / (br_elem * 1.0);
}

int main(int argc, char **argv)
{
   MPI_Init(NULL, NULL);
   int i = 0;
   int world_size;
   MPI_Comm_size(MPI_COMM_WORLD, &world_size);
   int world_rank;
   MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
   double elementi_za_sortiranje[BR_ELEMENTI];
   double elem_po_proces = BR_ELEMENTI / world_size;

   if (world_rank == 0)
   {
      for (i = 0; i < BR_ELEMENTI; i++)
      {
         elementi_za_sortiranje[i] = i;
      }
   }

   double *niza_po_proces = (double *)malloc(sizeof(double) * elem_po_proces);

   MPI_Scatter(elementi_za_sortiranje, elem_po_proces, MPI_DOUBLE, niza_po_proces,
elem_po_proces, MPI_DOUBLE, 0, MPI_COMM_WORLD);

   double avg = najdi_avg(niza_po_proces, elem_po_proces);

   printf("[%d]: Lokalen prosek:\t %.2f\n", world_rank, avg);
   double *lokalni_avg = NULL;
```

```
    if (world_rank == 0)
    {
        lokalni_avg = (double *)malloc(sizeof(double) * world_size);
    }
    MPI_Gather(&avg, 1, MPI_DOUBLE, lokalni_avg, 1, MPI_DOUBLE, 0,
MPI_COMM_WORLD);
    if (world_rank == 0)
    {
        printf("Globalen prosek:\t%.2f\n", najdi_avg(lokalni_avg, world_size));
    }
}
```

3. **Со користење на barrier, напишете програма којашто ќе стартува четири процеси. За секој од нив поставете бариера. Измерете го времето (користете ја clock() функцијата или пак MPI_Wtime()), и по преминот после бариерата за сите процеси испишете го времето на чекање на секој процес.**

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

int main(int argc, char **argv)
{
    MPI_Init(NULL, NULL);
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    double local_mpi_barrier_time = 0.0;
    MPI_Barrier(MPI_COMM_WORLD);
    double startTime = MPI_Wtime();
    double localTime = MPI_Wtime() - startTime;

    printf("[%d]\tWaited\t%.20f (time units)\n", rank, localTime);

    MPI_Finalize();
    return 0;
}
```