Паралелно и дистрибуирано процесирање

Лабораториска вежба 3

Петар Атанасовски - 216052

1. **Со користење на MPI_Isend и MPI_Irecv, да се напише програма со два процеси при што првиот процес ќе испрати порака на вториот, а вториот процес ќе врати порака на првиот.**

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

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

   int world_size;
   MPI_Comm_size(MPI_COMM_WORLD, &world_size);

   int rank;
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);

   const int BUFFER_SIZE = 128;
   char rec[BUFFER_SIZE];
   char send[BUFFER_SIZE];

   sprintf(send, "Message from CPU %d to CPU %d", rank, 1 - rank);

   MPI_Request reqs[2];
   MPI_Status statuses[2];

   MPI_Irecv(rec, BUFFER_SIZE, MPI_CHAR, 1 - rank, 0, MPI_COMM_WORLD, &reqs[0]);
   MPI_Isend(send, BUFFER_SIZE, MPI_CHAR, 1 - rank, 0, MPI_COMM_WORLD,
&reqs[1]);

   MPI_Waitall(2, reqs, statuses);

   printf("[%d]: %s\n", rank, rec);

   MPI_Finalize();
   return 0;
}
```

2. **Направете програма којашто ќе работи со n процеси (n > 3) . Користејќи ги MPI_Isend и MPI_Irecv како и соодветните техники за чекање на комплетирањето на овие операции, нека процесот со ранк 0 испрати пораки до сите останати процеси (1,...,n-1). Со извршување на соодветните проверки,**

по пристигнувањето на пораката во соодветниот процес (Било кој од 1,...n-1) испратете повратна порака до процесот со ранк 0.

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

#define BUFFER_SIZE 100
#define REQUEST_SIZE 20

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

   int world_size;
   MPI_Comm_size(MPI_COMM_WORLD, &world_size);

   int rank;
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);

   MPI_Request reqs[REQUEST_SIZE];
   MPI_Status statuses[REQUEST_SIZE];

   char recv[BUFFER_SIZE];
   char send[BUFFER_SIZE];

   int j = 0;

   if (rank == 0) {
      printf("[%d] : Sending messages to other CPUs\n", rank);
      strcpy(send, "Message from CPU0 to all CPUs");
      for (int proc_i = 1; proc_i < world_size; ++proc_i) {
         MPI_Isend(send, strlen(send) + 1, MPI_BYTE, proc_i, 0, MPI_COMM_WORLD,
&reqs[proc_i]);
      }
      MPI_Irecv(recv, BUFFER_SIZE, MPI_BYTE, world_size - 1, 0, MPI_COMM_WORLD,
&reqs[0]);
      MPI_Wait(&reqs[0], MPI_STATUS_IGNORE);
      printf("[%d] : Last message received: %s\n", rank, recv);
      j = world_size;
   } else {
      MPI_Irecv(recv, BUFFER_SIZE, MPI_BYTE, 0, 0, MPI_COMM_WORLD, &reqs[0]);
      MPI_Wait(&reqs[0], MPI_STATUS_IGNORE);
      printf("[%d] : Received message: %s\n", rank, recv);
      sprintf(send, "Answer from #%d to CPU0", rank);
      MPI_Isend(send, strlen(send) + 1, MPI_BYTE, 0, 0, MPI_COMM_WORLD, &reqs[j]);
      j++;
   }
```

```
    MPI_Waitall(j, reqs, statuses);

    MPI_Finalize();
    return 0;
}
```