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;

}

* 1. **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;

}