

Računarski sistemi visokih performansi

Nikola Vukić, Petar Trifunović, Veljko Petrović

Računarske vežbe
Zimski semestar 2023/24.

MPI 1

Sadržaj

- Šta je *MPI*, a šta *OpenMPI*?
- Kompajliranje i izvršavanje *MPI* programa.
- Komunikatori.
- *MPI* tipovi podataka.
- Point-to-Point komunikacija.
- Neblokirajuća Point-to-Point komunikacija.
- Dinamička Point-to-Point komunikacija.

Šta je *MPI*, a šta *OpenMPI*?

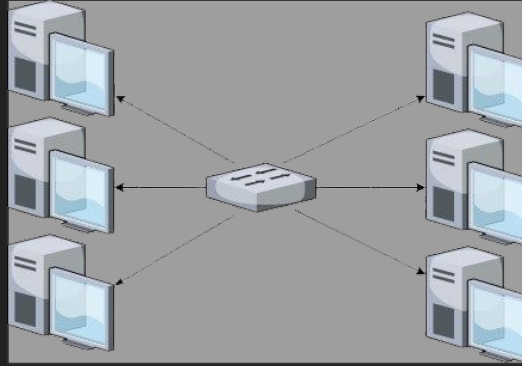
Message Passing Interface (MPI)

- Standard koji propisuje komunikaciju razmenom poruka na različitim paralelnim arhitekturama.
- Za održavanje standarda zadužen je MPI Forum (<https://www.mpi-forum.org/>).
- Trenutno važeća verzija standarda je verzija 4.1.
- Planira se i verzija 5.0.

Message Passing Interface (MPI)

- Postoje različite implementacije *MPI* standarda (komercijalne i otvorenog koda).
- Na ovom predmetu radi se *OpenMPI*, jedna od implementacija otvorenog koda.
- Postoji podrška za **C**, **C++** i *Fortran*.
- Postoje *binding* verzije za druge programske jezike (*wrapper*-i oko postojećih implementacija za **C**, **C++** i *Fortran*).
- *Binding* verzije čine *MPI* dostupnim u jezicima *C#*, *Java*, *R*, *Python*, *Ruby*, *Go*...

Ciljna arhitektura



- Računari (procesi) povezani mrežom preko koje komuniciraju radi izvršavanja posla u paraleli.
- Posao se po računarima (procesima) raspodeljuje razmenom poruka.

OpenMP i *MPI* — razlike

OpenMP	MPI
Skup kompajlerskih direktiva, bibliotečkih rutina i promenljivih okruženja	Standard koji ima različite implementacije
Paralelizacija na nivou niti	Paralelizacija na nivou procesa
Namenjen za sisteme sa deljenom memorijom	Namenjen za sisteme sa distribuiranom memorijom
Komunikacija preko deljene memorije	Komunikacija razmenom poruka preko mreže

OpenMPI

- Implementacija *MPI* standarda.
- Otvorenog koda.
- Implementacija za *C*, *C++* i *Fortran*.
- <https://github.com/open-mpi/ompi>

Format programa

```
#include <mpi.h>

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

    // MPI Code...

    MPI_Finalize();

    return 0;
}
```

Format programa

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



C biblioteka za *OpenMPI*

```
int main(int argc, char* argv[])  
{  
    MPI_Init(&argc, &argv);  
  
    // MPI Code...  
  
    MPI_Finalize();  
  
    return 0;  
}
```

Format programa

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

```
int main(int argc, char* argv[])  
{
```

```
    MPI_Init(&argc, &argv);
```

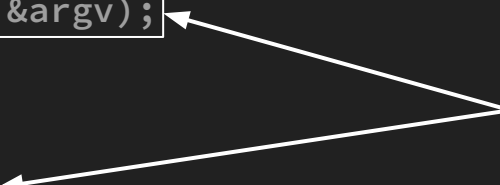
```
    // MPI Code...
```

```
    MPI_Finalize();
```

```
    return 0;
```

```
}
```

Neophodni pozivi za inicijalizaciju i
završetak *MPI* programa



Kompajliranje i izvršavanje *MPI* programa

Kompajliranje i izvršavanje *MPI* programa

- Kompajliranje:
 - `$ mpicc <izvorna_datoteka>`
- *mpicc* je omotačka skripta za *gcc*, pa se mogu koristiti i opcije *gcc* kompajlera.
- Pokretanje:
 - `$ mpiexec [-np N] [--bind-to core | hwthread] <izvršna_datoteka>`
 - `-np <N>` — opcija za zadavanje broja procesa koji će se kreirati pri pokretanju programa
 - `--bind-to` — opcija kojom se broj procesa koji će biti kreirani pri pokretanju programa vezuje ili za broj fizičkih (*core*), ili za broj logičkih (*hwthread*) jezgara procesora

Kompajliranje i izvršavanje *MPI* programa

- Kombinacija *-np <N>* i *--bind-to* opcija:
 - *--bind-to core* — ako je *<N>* manje ili jednako broju **fizičkih** jezgara, biće pokrenuto *<N>* procesa; ako je *<N>* veće, doći će do greške
 - *--bind-to hwthreads* — isto kao *core* opcija, samo se *<N>* upoređuje sa brojem **logičkih** jezgara
 - *-np <N>* bez *--bind-to* daće grešku ukoliko *<N>* bude veće od broja **fizičkih** jezgara
- *--oversubscribe* — u kombinaciji sa *-np <N>* uklanja ograničenja za vrednost parametra *<N>*
- Bez dodatnih opcija, pokrenuće se broj procesa jednak broju fizičkih jezgara.

Hello, world!

```
...  
int size, rank;  
  
MPI_Init(&argc, &argv);  
MPI_Comm_size(MPI_COMM_WORLD, &size);  
MPI_Comm_rank(MPI_COMM_WORLD, &rank);  
  
printf("Hello World iz %d/%d.\n", rank, size);  
...
```

- *primeri/p01_mpi_hello_world.c*

Hello, world!

```
...  
int size, rank;  
  
MPI_Init(&argc, &argv);  
MPI_Comm_size(MPI_COMM_WORLD, &size);  
MPI_Comm_rank(MPI_COMM_WORLD, &rank);  
printf("Hello World iz %d/%d.\n", rank, size);  
...
```

pribavljanje veličine komunikatora i
ranka procesa u komunikatoru (više
detalja o ovim funkcijama u nastavku)

- *primeri/p01_mpi_hello_world.c*

Kompajliranje i izvršavanje *MPI* programa

- Sve funkcije iz *mpi.h* biblioteke vraćaju celobrojni kôd greške.
- Povratna vrednost se može uporediti sa konstantom *MPI_SUCCESS*.
- Ako je povratna vrednost jednaka *MPI_SUCCESS*, nije nastupila greška.

Osnovni *MPI* koncepti

- Komunikator (engl. *communicator*)
- *Point-to-Point* komunikacija
- Kolektivna komunikacija

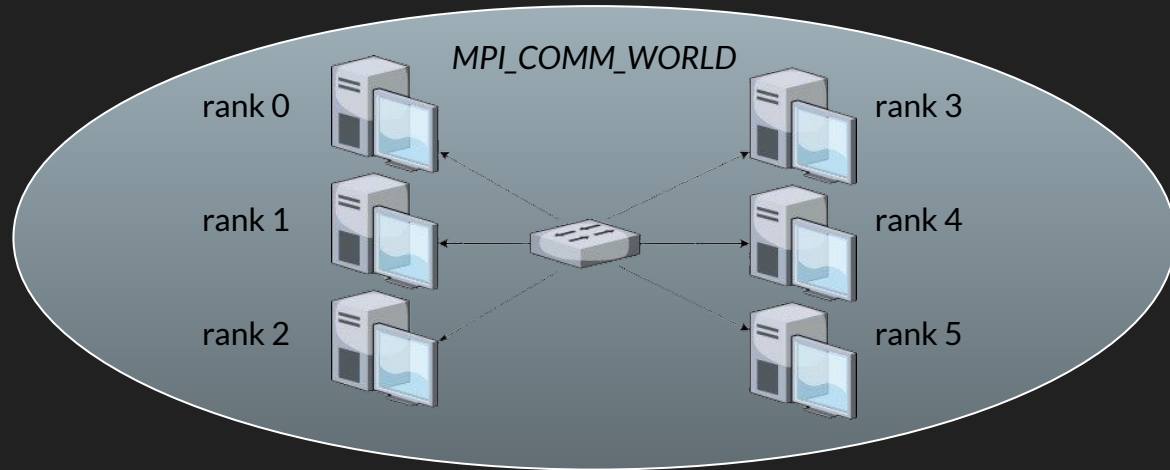
Komunikatori

Komunikatori

- Komunikator je grupa procesa unutar koje svaki proces ima svoj jedinstveni rank (odnosno identifikator).
- U C-u, promenljiva može biti tipa *MPI_Comm*, što je specijalna struktura kojom se predstavlja komunikator.
- Postoji nekoliko unapred definisanih komunikatora.

Predefinisani komunikatori — *MPI_COMM_WORLD*

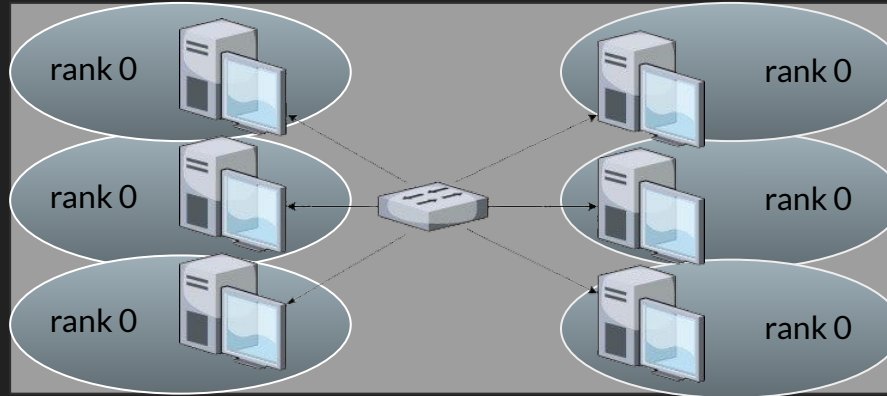
- Komunikator u kome se nalaze svi procesi pokrenutkog *MPI* programa.



Predefinisani komunikatori — *MPI_COMM_SELF*

- Svaki proces je jedini proces u svom *MPI_COMM_SELF* komunikatoru

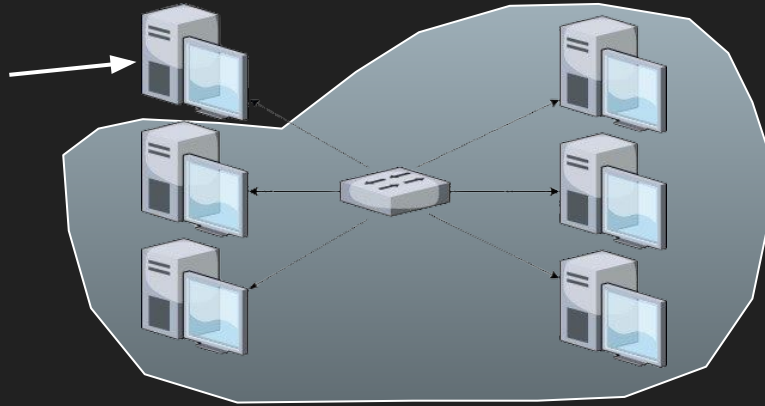
MPI_COMM_SELF
komunikatori svih procesa



Predefinisani komunikatori — *MPI_COMM_NULL*

- Bilo koji komunikator kome proces ne pripada.

Iz tačke gledišta ovog procesa, prikazani komunikator ima vrednost *MPI_COMM_NULL*.



Komunikatori — veličina i rank

- Veličina komunikatora ogleda se u broju procesa koji mu pripadaju.
- Funkcija za pribavljanje veličine komunikatora:

```
int MPI_Comm_size(MPI_Comm comm, int* size)
```

- Funkcija za pribavljanje ranka procesa unutar komunikatora:

```
int MPI_Comm_rank(MPI_Comm comm, int* rank)
```

Komunikatori — kreiranje

- Tokom izvršavanja *MPI* programa istovremeno može da postoji više komunikatora.
- *MPI_Comm* tip podataka.
- Neke funkcije za pravljenje komunikatora:

- deljenje postojećeg komunikatora

```
int MPI_Comm_split(MPI_Comm comm, int color, int key, MPI_Comm *newcomm)
```

- kopiranje postojećeg komunikatora

```
int MPI_Comm_dup(MPI_Comm comm, MPI_Comm *newcomm)
```

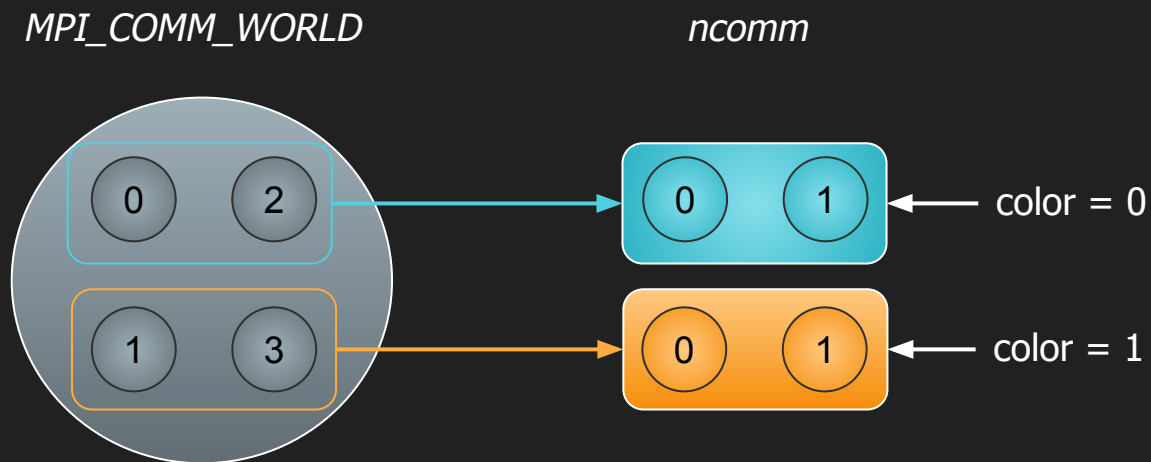
Komunikatori — *MPI_Comm_split*

- Deli prosleđeni komunikator na onoliko novih koliko ima različitih vrednosti parametra *color* među svim procesima.
- Parametar *color* — svi procesi koji pozovu *MPI_Comm_split* sa istom vrednošću ovog parametra naći će se u istom rezultujućem komunikatoru.
- Parametar *key* — procesi će u novom komunikatoru biti rangirani na osnovu vrednosti ovog parametra; ako dva procesa iste boje (*color*) imaju istu vrednost parametra *key*, rank će se odrediti na osnovu ranka iz komunikatora koji se deli.

Zadatak 1 — Deljenje komunikatora

- Napisati *OpenMPI* C program koji korišćenjem funkcije *MPI_Comm_split* na osnovu podrazumevanog pravi dva nova komunikatora. Procese podeliti u dva komunikatora na osnovu parnosti ranka unutar *MPI_COMM_WORLD* komunikatora. Pritom relativni poredak procesa unutar komunikatora treba da bude isti kao i u podrazumevanom komunikatoru. Svaki proces na standardni izlaz treba da ispiše svoj rank unutar *MPI_COMM_WORLD* i novoformiranog komunikatora ``ncomm``.
- Primer ispisa za jedan proces:
`MPI_COMM_WORLD rank: 0/4 - ncomm rank: 0/2`

Zadatak 1 — Ilustracija rešenja



- Primer rešenja: datoteka *resenja/01_mpi_communicators.c*.

MPI tipovi podataka

MPI tipovi podataka

- Zarad portabilnosti, MPI standard definiše tipove podataka.

<i>MPI tip podatka</i>	<i>C tip podatka</i>
<i>MPI_CHAR</i>	<i>char</i>
<i>MPI_SHORT</i>	<i>short int</i>
<i>MPI_INT</i>	<i>int</i>
<i>MPI_LONG</i>	<i>long int</i>
<i>MPI_UNSIGNED_CHAR</i>	<i>unsigned char</i>
<i>MPI_UNSIGNED_SHORT</i>	<i>unsigned short int</i>
<i>MPI_UNSIGNED</i>	<i>unsigned int</i>
<i>MPI_UNSIGNED_LONG</i>	<i>unsigned long int</i>

MPI tipovi podataka

- Zarad portabilnosti, MPI standard definiše tipove podataka.

<i>MPI tip podatka</i>	<i>C tip podatka</i>
<i>MPI_FLOAT</i>	<i>float</i>
<i>MPI_DOUBLE</i>	<i>double</i>
<i>MPI_LONG_DOUBLE</i>	<i>long double</i>
<i>MPI_BYTE</i>	
<i>MPI_PACKED</i>	

- Kompletna lista dostupna je u opisu standarda:

<https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report.pdf>

Point-to-Point komunikacija

Point-to-Point komunikacija

- Predstavlja komunikaciju između dva procesa.
- U osnovi, implementira se funkcijama za slanje i primanje poruke.
- U tim funkcijama, jasno se naglašava kom procesu se poruka šalje i od kog procesa se poruka prima.

Point-to-Point komunikacija

- Funkcije za razmenu poruka — slanje:

```
int MPI_Send(  
    const void *buf,  
    int count,  
    MPI_Datatype datatype,  
    int dest,  
    int tag,  
    MPI_Comm comm);
```

- Proces koji pozove ovu funkciju šalje *count* podataka tipa *datatype* sa memorijske lokacije *buf* procesu koji u komunikatoru *comm* ima rank *dest*.

Point-to-Point komunikacija

- Funkcije za razmenu poruka — prijem:

```
int MPI_Recv(  
    void *buf,  
    int count,  
    MPI_Datatype datatype,  
    int source,  
    int tag,  
    MPI_Comm comm,  
    MPI_Status *status);
```

- Proces koji pozove ovu funkciju prima maksimalno *count* podataka (a možda i manje) tipa *datatype* na memorijsku lokaciju *buf* od procesa koji u komunikatoru *comm* ima rank *source*.
- Promenljiva *status* sadrži informacije o primljenoj poruci; ukoliko nije od značaja, ovde se može proslediti *MPI_STATUS_IGNORE*.

Point-to-Point komunikacija

- Da bi proces *A* (poziva *MPI_Send*) uspešno poslao poruku procesu *B* (poziva *MPI_Recv*), potrebno je:
 - da *comm* parametar oba procesa ima istu vrednost,
 - da *dest* bude jednako ranku procesa *B*, a *source* ranku procesa *A* ili *MPI_ANY_SOURCE*, i
 - da *tag* ima istu vrednost u obe funkcije, ili da barem u jednoj bude *MPI_ANY_TAG*.

Point-to-Point komunikacija — primer

```
...
    if (rank == 0) {
        int message = 1;
        printf("Proces %d šalje poruku procesu %d.\n", rank, 1);
        MPI_Send(&message, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
    } else if (rank == 1) {
        int message;
        printf("Proces %d treba da primi poruku od procesa %d.\n", rank, 0);
        MPI_Recv(&message, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Proces %d primio poruku %d od procesa %d.\n", rank, message, 0);
    }
...

```

- `primeri/p02_mpi_send_recv.c`

Point-to-Point komunikacija — primer

```
...
if (rank == 0) {
    int message = 1;
    printf("Proces %d salje poruku procesu %d.\n", rank, 1);
    MPI_Send(&message, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
} else if (rank == 1) {
    int message;
    printf("Proces %d treba da primi poruku od procesa %d.\n", rank, 0);
    MPI_Recv(&message, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("Proces %d primio poruku %d od procesa %d.\n", rank, message, 0);
}
...
```

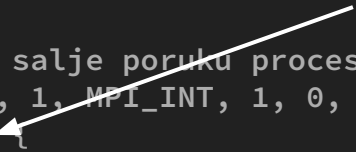
← obezbeđujemo da samo proces 0 šalje poruku

- `primeri/p02_mpi_send_recv.c`

Point-to-Point komunikacija — primer

```
...
if (rank == 0) {
    int message = 1;
    printf("Proces %d salje poruku procesu %d.\n", rank, 1);
    MPI_Send(&message, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
} else if (rank == 1) {
    int message;
    printf("Proces %d treba da primi poruku od procesa %d.\n", rank, 0);
    MPI_Recv(&message, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("Proces %d primio poruku %d od procesa %d.\n", rank, message, 0);
}
...
```

obezbeđujemo da samo proces 1 prima poruku




- *primeri/p02_mpi_send_recv.c*

Point-to-Point komunikacija — primer

proces 0 šalje jedan `MPI_INT` podatak sa adrese `&message` procesu sa rankom 1 u komunikatoru `MPI_COMM_WORLD`

```
...
if (rank == 0) {
    int message = 1;
    printf("Proces %d šalje poruku procesu %d.\n", rank, 1);
    MPI_Send(&message, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
} else if (rank == 1) {
    int message;
    printf("Proces %d treba da primi poruku od procesa %d.\n", rank, 0);
    MPI_Recv(&message, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("Proces %d primio poruku %d od procesa %d.\n", rank, message, 0);
}
...
```

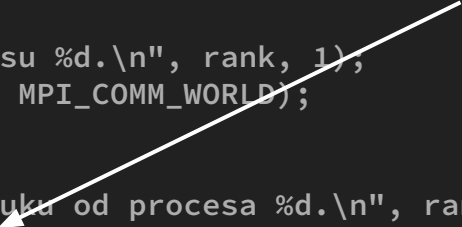


- `primeri/p02_mpi_send_recv.c`

Point-to-Point komunikacija — primer

proces 1 prima jedan *MPI_INT* podatak na adresi *&message* od procesa sa rankom 0 u komunikatoru *MPI_COMM_WORLD*

```
...
if (rank == 0) {
    int message = 1;
    printf("Proces %d salje poruku procesu %d.\n", rank, 1);
    MPI_Send(&message, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
} else if (rank == 1) {
    int message;
    printf("Proces %d treba da primi poruku od procesa %d.\n", rank, 0);
    MPI_Recv(&message, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("Proces %d primio poruku %d od procesa %d.\n", rank, message, 0);
}
...
```



- *primeri/p02_mpi_send_recv.c*

Point-to-Point komunikacija — modeli komunikacije

- Režimi slanja poruke:
 - *Synchronous* (*MPI_Ssend*) — pošiljalac šalje zahtev za slanje poruke, a poruka se šalje nakon što primalac odgovori na zahtev (*handshake* protokol); praktično, *MPI_Ssend* će se okončati samo ako postoji *MPI_Recv* sa kojim se može upariti.
 - *Buffered* (*MPI_Bsend*) — po iniciranju slanja, poruka se šalje u bafer odakle je primalac može preuzeti; ako nema mesta u baferu, doći će do greške.
 - *Standard* (*MPI_Send*) — može imati karakteristike *buffered*, ili *synchronous* režima; zavisi od veličine poruke, dostupnog prostora u baferu, i količine opzimizacije.
 - *Ready* (*MPI_Rsend*) — pretpostavlja se da proces primalac već čeka na poruku u trenutku iniciranja slanja i dolazi do greške ako to nije tačno; ovako je opisano u standardu, ali je u implementacijama ponašanje nedefinisano i ne preporučuje se korišćenje.
- Za prijem postoji samo jedan režim i poruka se smatra primljenom kada je preuzeta i može dalje da se koristi.

Napomena: “bafer” predstavlja sistemski bafer u koji se, u zavisnosti od mogućnosti i režima slanja, smeštaju podaci između pošiljaoca i primaoca

Point-to-Point — (ne)blokirajuća komunikacija

- Slanje i prijem poruke mogu biti:
 - blokirajući — Ako je slanje blokirajuće, kontrola toka se neće vratiti pozivaocu funkcije sve dok uslov slanja ne bude ispunjen. Nakon izlaska iz funkcije, bafer poruke može biti bezbedno prepisan. Ako je prijem blokirajuć, kontrola se ne vraća pozivaocu funkcije sve dok poruka ne bude preuzeta (**podrazumevano**).
 - neblokirajući — Iz funkcije slanja, tj. prijema poruke se izlazi nakon iniciranja slanja, tj. primanja poruke. Kada se pojavi potreba za korišćenjem izvornog, odnosno odredišnog bafera, potrebno je prethodno proveriti da li je podatak poslat, tj. primljen.

Napomena: “bafer” označava memorijsku lokaciju sa koje se šalju ili na koju se primaju podaci, odnosno prvi parametar *Send* i *Recv* funkcija

Neblokirajuća *Point-to-Point* komunikacija

Point-to-Point — neblokirajuća komunikacija

- Imena neblokirajućih funkcija:
 - *MPI_I[s, b, r]send(...), MPI_Irecv(...)*
- Dodatno slovo *I* od engleskog *Immediate*.
- Sve funkcije imaju iste parametre kao i blokirajuće varijante, uz dodatni parametar na poslednjem mestu *MPI_Request* request*.
- Parametar *request* će biti popunjen odgovarajućim vrednostima nakon poziva funkcije.
- Ovaj parametar je neophodan za upotrebu funkcija koje ispituju uspešnost slanja i prijema.

Point-to-Point neblokirajuća komunikacija — ispitivanje stanja slanja i prijema

```
int MPI_Test(  
    MPI_Request *request,  
    int *flag,  
    MPI_Status *status)
```

- Testira stanje zahteva *request* i postavlja promenljivu *flag* na *true* ukoliko je zahtev završen, a na *false* ako nije.

```
int MPI_Wait(  
    MPI_Request *request,  
    MPI_Status *status)
```

- Blokira izvršavanje programa sve dok se zahtev *request* ne završi.

Point-to-Point neblokirajuća komunikacija — primer

```
... if (rank == 0) {  
    MPI_Request send_request;  
    char *message = "Zdravo!";  
  
    MPI_Issend(message, 8, MPI_CHAR, 1, 0, MPI_COMM_WORLD, &send_request);  
    printf("Proces %d inicirao slanje poruke.\n", rank);  
    printf("Proces %d radi nesto drugo dok se poruka salje...\n", rank);  
  
    int flag = 0;  
    MPI_Test(&send_request, &flag, MPI_STATUS_IGNORE);  
    if (flag != 0) {  
        // poslato  
    } else {  
        // još uvek nije poslato  
    }  
} else ...
```

- *primeri/p03_mpi_send_recv_nonblocking.c*

Point-to-Point neblokirajuća komunikacija — primer

```
... if (rank == 0) {  
    ...  
} else {  
    MPI_Request receive_request;  
    char message[8];  
    MPI_Irecv(message, 8, MPI_CHAR, 0, 0, MPI_COMM_WORLD, &receive_request);  
    printf("Proces %d inicirao primanje poruke.\n", rank);  
  
    printf("Proces %d radi nesto drugo dok se poruka prima...\n", rank);  
  
    MPI_Wait(&receive_request, MPI_STATUS_IGNORE);  
    printf("Proces %d primio poruku: \"%s\"\n", rank, message);  
}
```

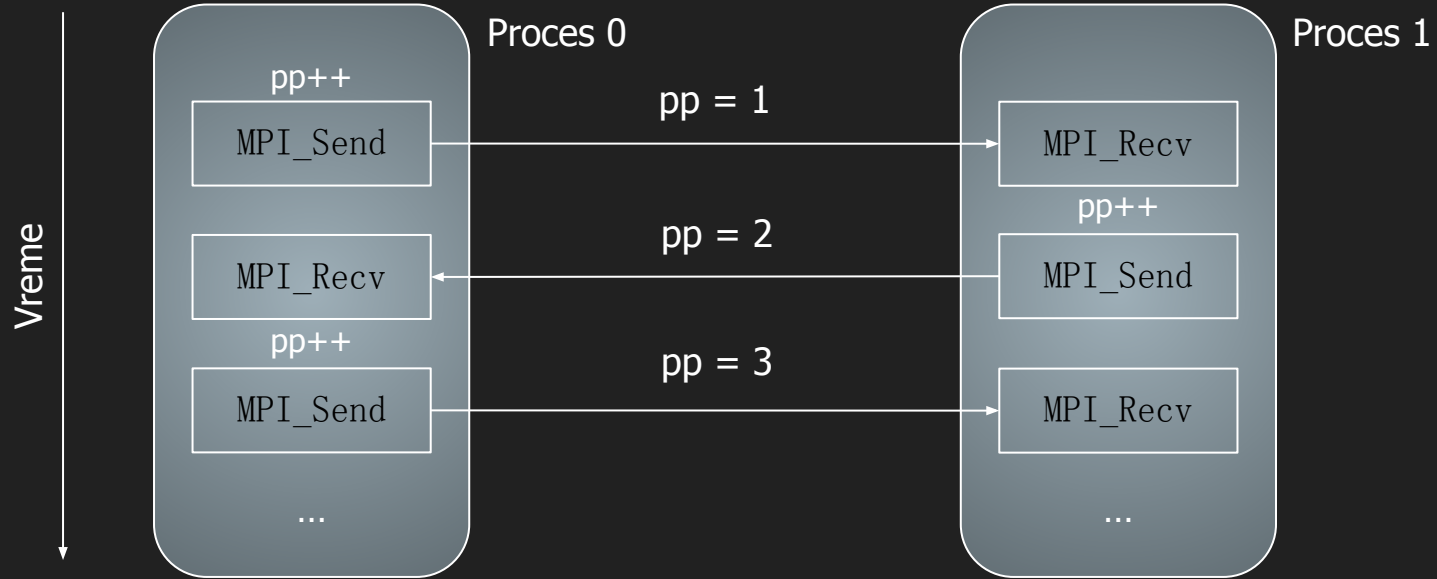
- *primeri/p03_mpi_send_recv_nonblocking.c*

Zadatak 2 — Ping-pong

- Napisati *OpenMPI* C program koji simulira igranje ping-ponga između dva procesa. Lopticu simulirati promenljivom tipa *int*. Uvećati ovu promenljivu svaki put kada neki od procesa udari lopticu, odnosno, pre nego što proces pošalje promenljivu onom drugom, i ispisati odgovarajuću poruku.
- Primer očekivanog ispisa:

```
p0 sent ping_pong_count to p1 and incremented it to 1.  
p1 received ping_pong_count 1 from p0.  
p1 sent ping_pong_count to p0 and incremented it to 2.
```
- Napomene:
 - Pretpostavka je da će program biti pozvan za *-np 2* opcijom i to ne treba proveravati.
 - Ne treba smatrati da je program neispravan ako ispis ne bude u ispravnom redosledu.
- Primer rešenja: datoteka *resenja/02_mpi_ping_pong.c*.

Zadatak 2 — Ping-pong



`pp` — Promenljiva kojom se simulira loptica.

Zadatak 3 — Sekvencijalni ping-pong

- Modifikovati ping-pong zadatak tako da se ispis na standardni izlaz odvija u redosledu u kom procesi udaraju ping-pong lopticu. Program pokrenuti sa tri procesa — procesi 0 i 1 igraju ping-pong, dok treći proces ispisuje poruke na standardni izlaz. Svaki put kada neki od procesa igrača udari lopticu, on procesu štampaču šalje poruku koju treba ispisati na standardni izlaz. Poruke za ispis procesu štampaču stižu u proizvoljnom redosledu, ali on treba da ih ispiše u redosledu koji odgovara sekvencijalnom izvršavanju programa. Sve poruke su iste dužine. Ping-pong se igra do 9.
- Primer očekivanog ispisa:

```
p0 sent ping_pong_count to p1 and incremented it to 1.  
p1 sent ping_pong_count to p0 and incremented it to 2.  
p0 sent ping_pong_count to p1 and incremented it to 3.
```
- Primer rešenja: datoteka *resenja/03_mpi_ping_pong_seq.c*.

Zadatak 3 — Sekvencijalni ping-pong

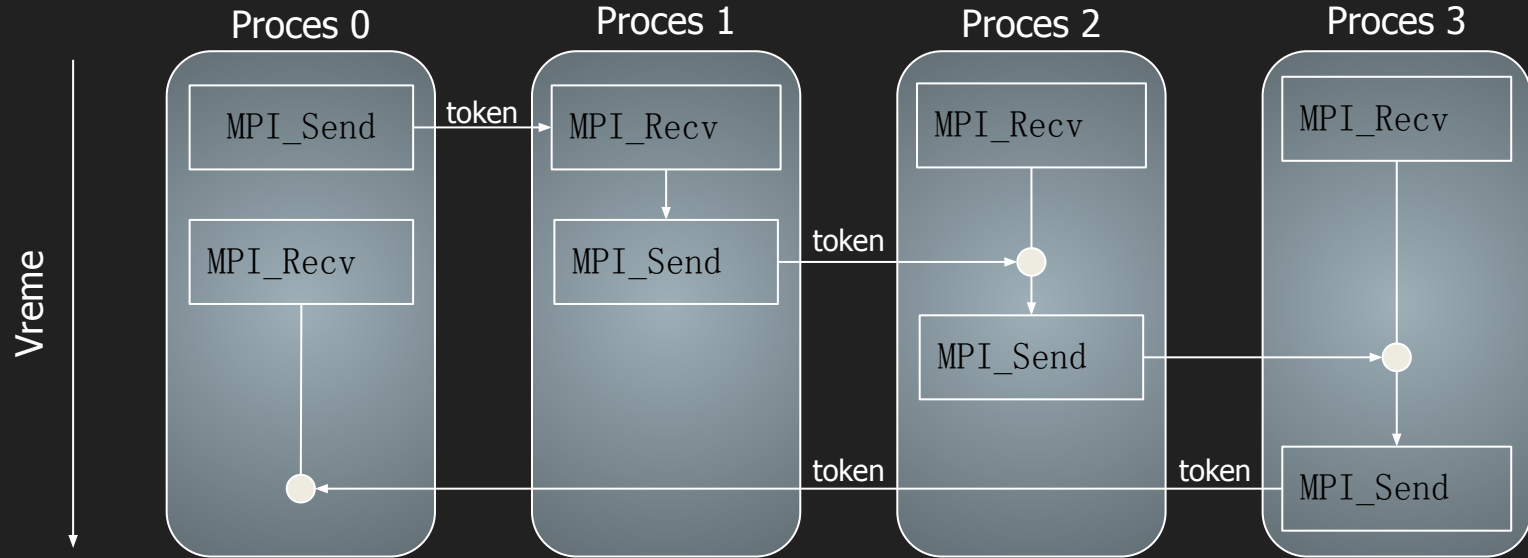
- Pojašnjenje rešenja:
 - Polje *tag* u procesu štampaču (**proces 2**) obezbeđuje da se ispisuju poruke u ispravnom redosledu, bez obzira na to koja je kad poslata.
 - Ako se za slanje koristi *MPI_Send*, a u pozadini se zapravo iskoristi sinhroni režim, proces koji šalje poruku neće preći u narednu iteraciju *while* petlje sve dok proces štampač ne pozove *MPI_Recv*. Ovo može (minimalno) usporiti rad programa.
 - Nešto bolje performanse dobiće se ukoliko se koristi *MPI_Bsend*.

Zadatak 4 — Prsten

- Napisati *OpenMPI* C program koji prosleđuje žeton između procesa po principu prtena. Žeton je predstavljen brojem -1 i poseduje ga proces ranga 0. Svi procesi osim poslednjeg šalju žeton procesu sa rangom za jedan većim od svog. Poslednji proces (proces sa najvećim rangom u komunikatoru) žeton prosleđuje nazad procesu ranga 0.
- Nakon što proces ranga 0 primi žeton, program se završava.
- Ispisati poruku na standardni izlaz svaki put kada neki od procesa primi žeton.
- Primer očekivanog ispisa:

```
Process 1 received token -1 from process 0
Process 2 received token -1 from process 1
Process 3 received token -1 from process 2
Process 0 received token -1 from process 3
```
- Primer rešenja: datoteka *resenja/04_mpi_ring.c*.

Zadatak 4 — Prsten



Zadatak 4 — Prsten

- Pojašnjenje rešenja:
 - Na početku svi pozivaju *MPI_Recv* osim procesa sa rankom nula.
 - Kada bi i **proces 0** pozvao funkciju za prijem poruke, došlo bi do *deadlock*-a, jer bi svi procesi čekali na poruku koju još uvek nije poslao.

Zadatak 5 — Ping-pong, neblokirajući sekvencijalni ispis

- Modifikovati zadatak 3 tako da slanje poruka procesu štampaču bude neblokirajuće. Pritom obezbediti da program radi korektno, odnosno da se ne desi da poruka koja nije poslata bude prepisana novom porukom pre nego što se stara pošalje.
- Primer rešenja: datoteka *resenja/05_mpi_ping_pong_printf_async.c*.

Zadatak 5 — Ping-pong, neblokirajući sekvencijalni ispis

- Pojašnjenje rešenja:
 - Da u rešenju nema *if(has_sent)* provere, poziv *MPI_Wait* bi, kada bi ostao na istom mestu u kodu, doveo do greške, jer *send_req* ne bi bila postavljena ni na koju vrednost.
 - Bez *has_sent* provere, da ne bi došlo do greške, *MPI_Wait* poziv bi morao da sledi odmah nakon *MPI_Isend*, da se promenljiva *send_str* ne bi promenila pre nego što zapravo bude poslata.
 - *MPI_Isend* poziv praćen pozivom *MPI_Wait* bez bilo kakvog izvršenja između ovih poziva imao bi isti efekat kao da je pozvana blokirajuća *Send* varijanta.
 - Iz ovog razloga je uvedena *has_sent* provera, koja omogućava da se pre *MPI_Wait* izvrši iteracija koja poziva *MPI_Recv*, kao i deo naredne iteracije koji prethodi *MPI_Wait* pozivu.

Zadatak 5 — Ping-pong, neblokirajući sekvencijalni ispis

- Pojašnjenje rešenja (iz ugla procesa 0):

```
while (ping_pong_count < PING_PONG_LIMIT) {  
    if (world_rank == ping_pong_count % 2) {  
        ...  
        if(has_sent) {  
            MPI_Wait(&send_req, MPI_STATUS_IGNORE);  
        }  
        MPI_Isend(send_str, strlen(send_str)+1, MPI_CHAR, 2, ping_pong_count, MPI_COMM_WORLD, &send_req);  
        has_sent = 1;  
    } else if (world_rank == (ping_pong_count + 1) % 2) {  
        MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD,  
                MPI_STATUS_IGNORE);  
    } else ...  
}
```

u prvoj iteraciji, *ping_pong_count* je 0,
pa proces 0 prolazi ovaj uslov

Zadatak 5 — Ping-pong, neblokirajući sekvencijalni ispis

- Pojašnjenje rešenja (iz ugla procesa 0):

```
while (ping_pong_count < PING_PONG_LIMIT) {  
    if (world_rank == ping_pong_count % 2) {  
        ...  
        if(has_sent) {  
            MPI_Wait(&send_req, MPI_STATUS_IGNORE);  
        }  
        MPI_Isend(send_str, strlen(send_str)+1, MPI_CHAR, 2, ping_pong_count, MPI_COMM_WORLD, &send_req);  
        has_sent = 1;  
    } else if (world_rank == (ping_pong_count + 1) % 2) {  
        MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD,  
                MPI_STATUS_IGNORE);  
    } else ...  
}
```

u ovom delu (skraćen zbog preglednosti), proces 0 će inkrementirati *ping_pong_count*

Zadatak 5 — Ping-pong, neblokirajući sekvencijalni ispis

- Pojašnjenje rešenja (iz ugla procesa 0):

```
while (ping_pong_count < PING_PONG_LIMIT) {  
    if (world_rank == ping_pong_count % 2) {  
        ...  
        if(has_sent) {  
            MPI_Wait(&send_req, MPI_STATUS_IGNORE);  
        }  
        MPI_Isend(send_str, strlen(send_str)+1, MPI_CHAR, 2, ping_pong_count, MPI_COMM_WORLD, &send_req);  
        has_sent = 1;  
    } else if (world_rank == (ping_pong_count + 1) % 2) {  
        MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD,  
            MPI_STATUS_IGNORE);  
    } else ...  
}
```

has_sent je 0, pa se MPI_Wait ne poziva

Zadatak 5 — Ping-pong, neblokirajući sekvencijalni ispis

- Pojašnjenje rešenja (iz ugla procesa 0):

```
while (ping_pong_count < PING_PONG_LIMIT) {  
    if (world_rank == ping_pong_count % 2) {  
        ...  
        if (has_sent) {  
            MPI_Wait(&send_req, MPI_STATUS_IGNORE);  
        }  
        MPI_Isend(send_str, strlen(send_str)+1, MPI_CHAR, 2, ping_pong_count, MPI_COMM_WORLD, &send_req);  
        has_sent = 1;  
    } else if (world_rank == (ping_pong_count + 1) % 2) {  
        MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD,  
            MPI_STATUS_IGNORE);  
    } else ...  
}
```

proces 0 šalje poruku bez blokiranja,
tako da odmah prelazi na sledeću
instrukciju i postavlja *has_sent* na 1

Zadatak 5 — Ping-pong, neblokirajući sekvencijalni ispis

- Pojašnjenje rešenja (iz ugla procesa 0):

```
while (ping_pong_count < PING_PONG_LIMIT) {  
    if (world_rank == ping_pong_count % 2) {  
        ...  
        if(has_sent) {  
            MPI_Wait(&send_req, MPI_STATUS_IGNORE);  
        }  
        MPI_Isend(send_str, strlen(send_str)+1, MPI_CHAR, 2, ping_pong_count, MPI_COMM_WORLD, &send_req);  
        has_sent = 1;  
    } else if (world_rank == (ping_pong_count + 1) % 2) {  
        MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD,  
                MPI_STATUS_IGNORE);  
    } else ...  
}
```

proces 0 sada može da pređe na sledeću iteraciju

Zadatak 5 — Ping-pong, neblokirajući sekvencijalni ispis

- Pojašnjenje rešenja (iz ugla procesa 0):

```
while (ping_pong_count < PING_PONG_LIMIT) {  
    if (world_rank == ping_pong_count % 2) {  
        ...  
        if(has_sent) {  
            MPI_Wait(&send_req, MPI_STATUS_IGNORE);  
        }  
        MPI_Isend(send_str, strlen(send_str)+1, MPI_CHAR, 2, ping_pong_count, MPI_COMM_WORLD, &send_req);  
        has_sent = 1;  
    } else if (world_rank == (ping_pong_count + 1) % 2) {  
        MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD,  
                MPI_STATUS_IGNORE);  
    } else ...  
}
```


u ovoj iteraciji, *ping_pong_count* u procesu 0 iznosi 1, pa će proces 0 preći u ovu granu izvršenja



Zadatak 5 — Ping-pong, neblokirajući sekvencijalni ispis

- Pojašnjenje rešenja (iz ugla procesa 0):

```
while (ping_pong_count < PING_PONG_LIMIT) {  
    if (world_rank == ping_pong_count % 2) {  
        ...  
        if(has_sent) {  
            MPI_Wait(&send_req, MPI_STATUS_IGNORE);  
        }  
        MPI_Isend(send_str, strlen(send_str)+1, MPI_CHAR, 2, ping_pong_count, MPI_COMM_WORLD, &send_req);  
        has_sent = 1;  
    } else if (world_rank == (ping_pong_count + 1) % 2) {  
        MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD,  
                MPI_STATUS_IGNORE);  
    } else ...  
}
```



nakon prijema poruke od procesa 1, proces 0
ponovo može preći na sledeću iteraciju

Zadatak 5 — Ping-pong, neblokirajući sekvencijalni ispis

- Pojašnjenje rešenja (iz ugla procesa 0):

```
while (ping_pong_count < PING_PONG_LIMIT) {  
    if (world_rank == ping_pong_count % 2) {  
        ...  
        if(has_sent) {  
            MPI_Wait(&send_req, MPI_STATUS_IGNORE);  
        }  
        MPI_Isend(send_str, strlen(send_str)+1, MPI_CHAR, 2, ping_pong_count, MPI_COMM_WORLD, &send_req);  
        has_sent = 1;  
    } else if (world_rank == (ping_pong_count + 1) % 2) {  
        MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD,  
                MPI_STATUS_IGNORE);  
    } else ...  
}
```

ping_pong_count u procesu 0 sada iznosi 2, pa će proces 0 proći ovaj uslov

Zadatak 5 — Ping-pong, neblokirajući sekvencijalni ispis

- Pojašnjenje rešenja (iz ugla procesa 0):

```
while (ping_pong_count < PING_PONG_LIMIT) {  
    if (world_rank == ping_pong_count % 2) {  
        ...  
        if(has_sent) ← has_sent je sada 1, pa će proces 0 proći ovaj uslov  
            MPI_Wait(&send_req, MPI_STATUS_IGNORE);  
        }  
        MPI_Isend(send_str, strlen(send_str)+1, MPI_CHAR, 2, ping_pong_count, MPI_COMM_WORLD, &send_req);  
        has_sent = 1;  
    } else if (world_rank == (ping_pong_count + 1) % 2) {  
        MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD,  
            MPI_STATUS_IGNORE);  
    } else ...  
}
```

Zadatak 5 — Ping-pong, neblokirajući sekvencijalni ispis

- Pojašnjenje rešenja (iz ugla procesa 0):

```
while (ping_pong_count < PING_PONG_LIMIT) {  
    if (world_rank == ping_pong_count % 2) {  
        ...  
        if(has_sent) {  
            MPI_Wait(&send_req, MPI_STATUS_IGNORE);  
        }  
        MPI_Isend(send_str, strlen(send_str)+1, MPI_CHAR, 2, ping_pong_count, MPI_COMM_WORLD, &send_req);  
        has_sent = 1;  
    } else if (world_rank == (ping_pong_count + 1) % 2) {  
        MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD,  
                MPI_STATUS_IGNORE);  
    } else ...  
}
```

proces 0 će se zaustaviti tek ovde, i to samo ako
poruka iz *MPI_Isend* poziva još uvek nije poslata

Zadatak 5 — Ping-pong, neblokirajući sekvencijalni ispis

- Pojašnjenje rešenja (iz ugla procesa 0):

```
while (ping_pong_count < PING_PONG_LIMIT) {  
    if (world_rank == ping_pong_count % 2) {  
        ...  
        if(has_sent) {  
            MPI_Wait(&send_req, MPI_STATUS_IGNORE);  
        }  
        MPI_Isend(send_str, strlen(send_str)+1, MPI_CHAR, 2, ping_pong_count, MPI_COMM_WORLD, &send_req);  
        has_sent = 1;  
    } else if (world_rank == (ping_pong_count + 1) % 2) {  
        MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD,  
                MPI_STATUS_IGNORE);  
    } else ...  
}
```

proces 0 se zaustavlja tek **ovde** u iteraciji 2

Zadatak 5 — Ping-pong, neblokirajući sekvencijalni ispis

- Pojašnjenje rešenja (iz ugla procesa 0):

```
while (ping_pong_count < PING_PONG_LIMIT) {  
    if (world_rank == ping_pong_count % 2) {  
        ...  
        if (has_sent) {  
            MPI_Wait(&send_req, MPI_STATUS_IGNORE);  
        }  
        MPI_Isend(send_str, strlen(send_str)+1, MPI_CHAR, 2, ping_pong_count, MPI_COMM_WORLD, &send_req);  
        has_sent = 1;  
    } else if (world_rank == (ping_pong_count + 1) % 2) {  
        MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD,  
                MPI_STATUS_IGNORE);  
    } else ...  
}
```

proces 0 se zaustavlja tek **ovde** u iteraciji 2

sa blokirajućim pozivom bi se potencijalno zaustavio već **ovde** u iteraciji 0

Dinamička *Point-to-Point* komunikacija

Point-to-Point — dinamička komunikacija

- Nekada poruke koje procesi razmenjuju nisu fiksne dužine.
- Tada je prvo potrebno očitati dužinu poruke, alocirati bafer za poruku, pa tek onda započeti njeno primanje.

Point-to-Point dinamička komunikacija

```
int MPI_Probe(  
    int source,  
    int tag,  
    MPI_Comm comm,  
    MPI_Status *status)
```

- Simulira primanje poruke.
- Popunjava *status* polje.

```
int MPI_Get_count(  
    const MPI_Status *status,  
    MPI_Datatype datatype,  
    int count)
```

- Na osnovu *status* polja, očitava broj podataka tipa *datatype* koji će biti primljeni.

Point-to-Point dinamička komunikacija — primer

```
if (rank == 0) {  
    int size = rand() % 10 + 1;  
    char *message = (char *) calloc(size + 1, sizeof(char));  
  
    for (int i = 0; i < size; i++) {  
        message[i] = 'a';  
    }  
  
    MPI_Send(message, size + 1, MPI_CHAR, 1, 0, MPI_COMM_WORLD);  
    free(message);  
} else if ...
```

- *primeri/p04_mpi_dynamic_communication.c*

Point-to-Point dinamička komunikacija — primer

```
if (rank == 0) {  
    ...  
} else if (rank == 1) {  
    MPI_Status status;  
    int size;  
    MPI_Probe(0, 0, MPI_COMM_WORLD, &status);  
    MPI_Get_count(&status, MPI_CHAR, &size);  
  
    printf("Velicina poruke u karakterima: %d\n", size);  
  
    char *message = (char *) malloc(size * sizeof(char));  
    MPI_Recv(message, size, MPI_CHAR, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);  
    printf("Primljena poruka: %s.\n", message);  
}
```

- `primeri/p04_mpi_dynamic_communication.c`

Zadatak 6 — Ping pong, promenljiva dužina poruke

- Modifikovati zadatak 3 tako da se procesu štampaču šalju poruke promenljive dužine.
- Koristiti funkcije *MPI_Probe* i *MPI_Get_count*.
- Procesi mogu da izvrše maksimalno 999 razmena lopticom.
- Primer rešenja: datoteka *resenja/06_mpi_ping_pong_printf_variablelen.c*.