



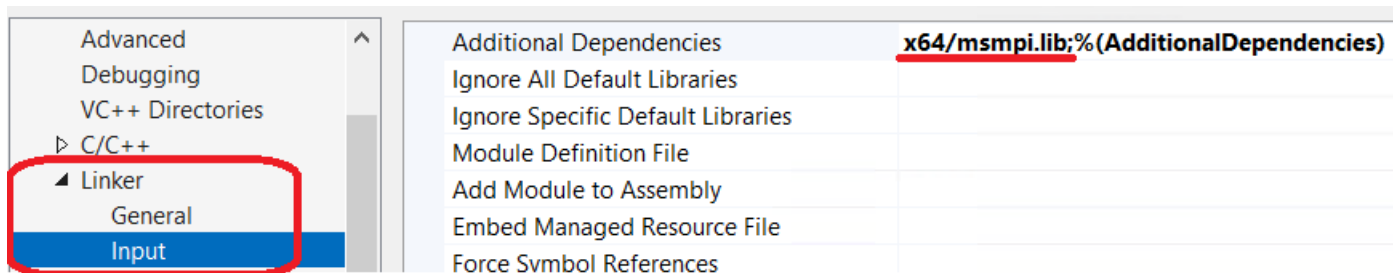
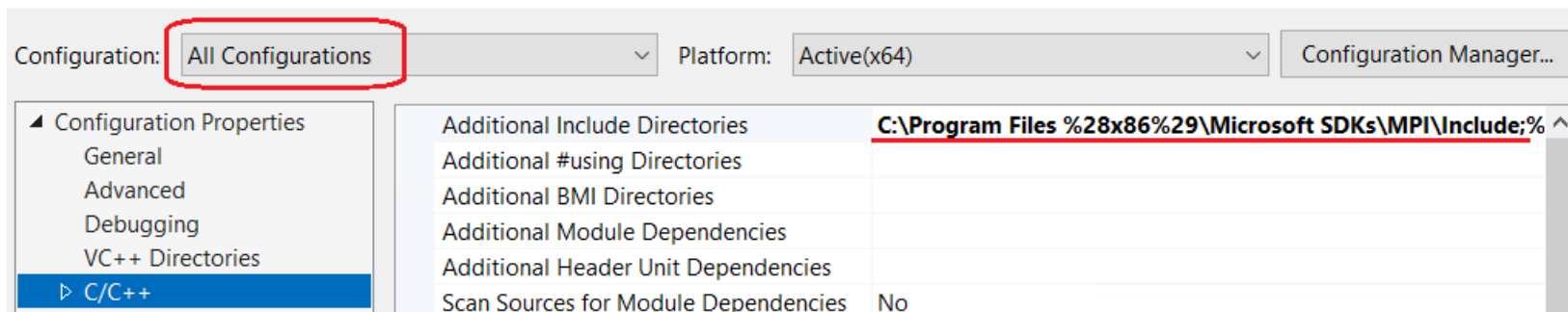
SVEUČILIŠTE U ZAGREBU
Fakultet
elektrotehnike i
računarstva

PARALELNO PROGRAMIRANJE

Poglavlje 02 - MPI

AG 2024/2025

VS POSTAVKE



HELLO WORLD

```
// inicijalizacija
```

```
MPI_Init(NULL, NULL);
```



Obavezno na početku!

```
// ukupan broj procesa
```

```
int world_size;
```

```
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
```



Ukupno procesa

```
// moj redni broj
```

```
int world_rank;
```

```
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
```



Vlastiti id

```
// ime procesora (CPU)
```

```
char processor_name[MPI_MAX_PROCESSOR_NAME];
```

```
int name_len;
```

```
MPI_Get_processor_name(processor_name, &name_len);
```

```
// hello world poruka
```

```
printf("Hello world from processor %s, rank %d"  
      " out of %d processors\n",  
      processor_name, world_rank, world_size);
```

```
// kraj programa
```

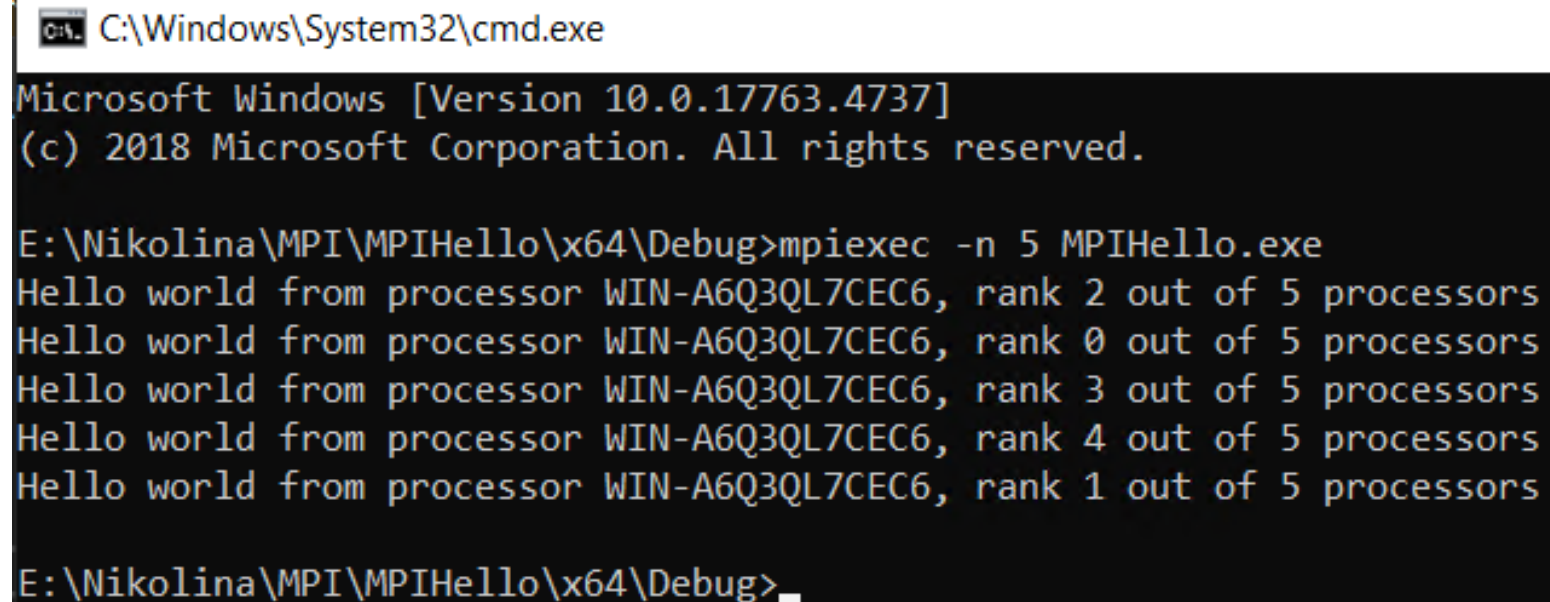
```
MPI_Finalize();
```



Obavezno na kraju!

HELLO WORLD

- `mpiexec -n 5 MPIHello.exe`



A screenshot of a Windows command prompt window. The title bar shows the path `C:\Windows\System32\cmd.exe`. The prompt text is `Microsoft Windows [Version 10.0.17763.4737]
(c) 2018 Microsoft Corporation. All rights reserved.`. The user has entered the command `E:\Nikolina\MPI\MPIHello\x64\Debug>mpiexec -n 5 MPIHello.exe`. The output shows five lines of "Hello world" messages, each from a different processor rank (2, 0, 3, 4, 1) out of 5 processors, all on the same machine (WIN-A6Q3QL7CEC6). The prompt ends with `E:\Nikolina\MPI\MPIHello\x64\Debug>`.

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.17763.4737]
(c) 2018 Microsoft Corporation. All rights reserved.

E:\Nikolina\MPI\MPIHello\x64\Debug>mpiexec -n 5 MPIHello.exe
Hello world from processor WIN-A6Q3QL7CEC6, rank 2 out of 5 processors
Hello world from processor WIN-A6Q3QL7CEC6, rank 0 out of 5 processors
Hello world from processor WIN-A6Q3QL7CEC6, rank 3 out of 5 processors
Hello world from processor WIN-A6Q3QL7CEC6, rank 4 out of 5 processors
Hello world from processor WIN-A6Q3QL7CEC6, rank 1 out of 5 processors

E:\Nikolina\MPI\MPIHello\x64\Debug>
```

MPI RAZMJENA PORUKA

```
...  
int number;  
  
//slanje u lancu, proc id=0 počinje, zatim čeka što će dobiti od n-1-og  
if (world_rank == 0) {  
    number = 0;
```

```
MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
```

```
MPI_Recv(&number, 1, MPI_INT, world_size - 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

```
}  
  
else{  
    MPI_Recv(&number, 1, MPI_INT, world_rank-1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);  
  
    number++;  
  
    MPI_Send(&number, 1, MPI_INT, (world_rank + 1) % world_size, 0, MPI_COMM_WORLD);
```

```
...
```

MPI RAZMJENA PORUKA

- `mpiexec -n 5 MPIComm.exe`

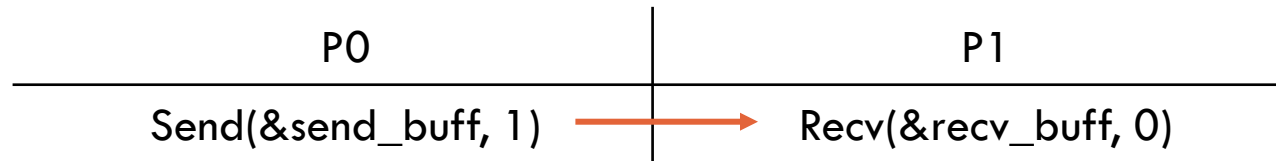
```
Microsoft Windows [Version 10.0.17763.4737]
(c) 2018 Microsoft Corporation. All rights reserved.

E:\Nikolina\MPI\MPIComm\x64\Debug>mpiexec -n 5 MPIComm.exe
Proces 0 salje broj 0 procesu 1
Proces 0 prima broj 4 od procesa 4
Proces 3 prima broj 2 od procesa 2
Proces 3 salje broj 3 procesu 4
Proces 2 prima broj 1 od procesa 1
Proces 2 salje broj 2 procesu 3
Proces 1 prima broj 0 od procesa 0
Proces 1 salje broj 1 procesu 2
Proces 4 prima broj 3 od procesa 3
Proces 4 salje broj 4 procesu 0

E:\Nikolina\MPI\MPIComm\x64\Debug>_
```

SKRAĆENA SINTAKSA

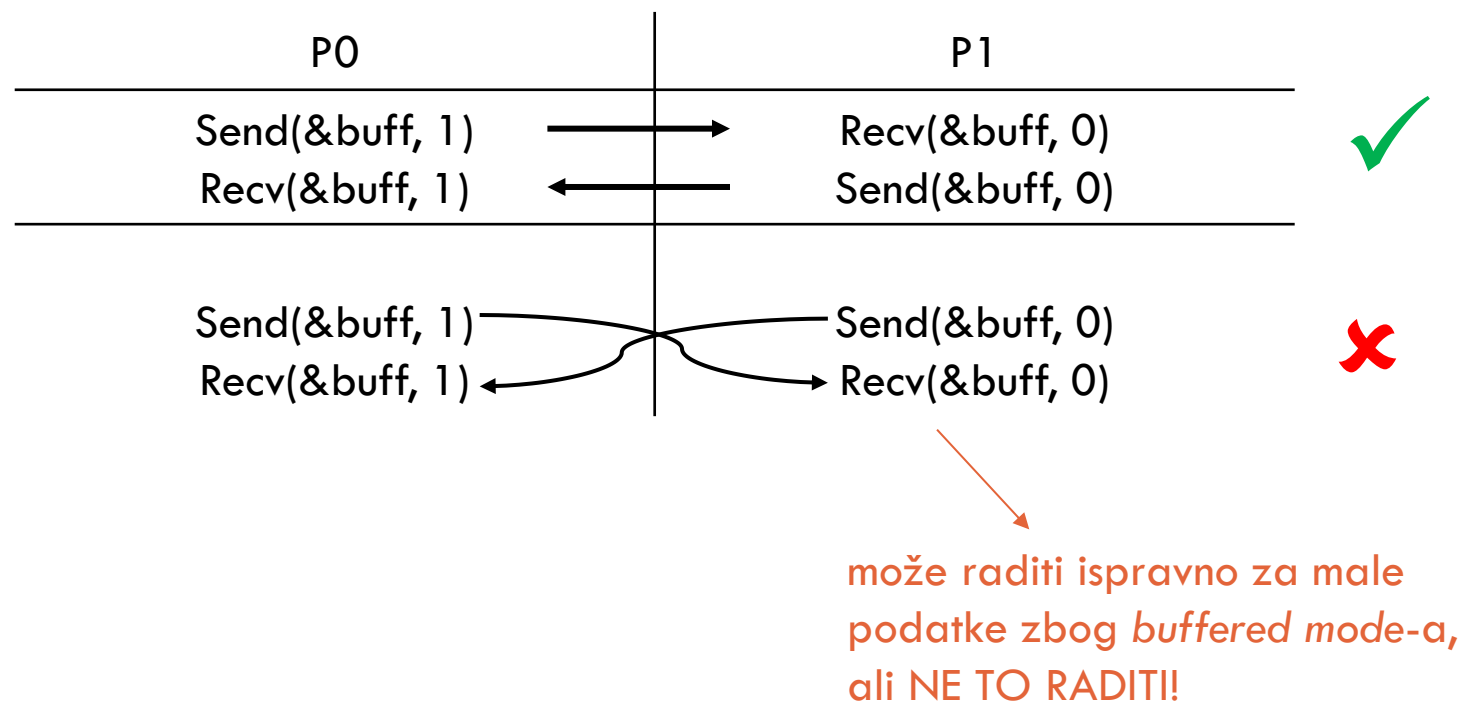
- Send (<što>, <kome>)
- Recv (<što>, <od koga>)
- Recv (<što>, *)



** oba procesa MORAJU sudjelovati u komunikaciji da bi ona uspjela (jedan poziva SEND, a drugi RECV)*

BLOKIRANJE

- SEND, RECV → blokirajuće funkcije



POOPĆENJE PRIMANJA

- MPI_ANY_SOURCE
 - pogodno kad nije bitan redoslijed primanja poruka
- MPI_ANY_TAG

**konkretni podaci o pošiljatelju i/ili tag-u se mogu pročitati iz parametra Status*

PRIMJER: izračun PI

→ CPI.cpp

→ ICPI.cpp

ZADATAK 1

Korištenjem MPI funkcija Send i Recv (skraćena sintaksa) napišite niz instrukcija koji će **sve elemente zadane kružne liste postaviti na srednju vrijednost toga i dvaju susjednih elemenata** (indeksi i , $i+1$, $i-1$; posljednji element povezan je s prvim i obrnuto).

Svaki MPI proces ima u lokalnoj memoriji samo jedan element liste koji je realna vrijednost. **Broj procesa je N** , svaki proces ima **redni broj ID**. Program treba jamčiti ispravnost rada bez obzira na veličinu poruka (ne smije doći do potpunog zastoja zbog redoslijeda slanja i primanja)!

ZADATAK 2

U jednom trenutku rada paralelnog programa u n procesa se nalaze neki podaci. Potrebno je odrediti **najveći element od svih n podataka** i tu informaciju (vrijednost najvećega) proslijediti svim procesima. Napisati algoritam koji će obaviti taj zadatak pomoću MPI funkcija `MPI_Send` i `MPI_Recv`.

Uputa: slanje i primanje poruka obaviti u obliku **lanca u dva prolaza** (s lijeva na desno te potom s desna na lijevo po svim procesima). Kod poziva MPI funkcija navesti samo 'bitne' parametre (npr. `MPI_Send(&varijabla, __, __, odrediste, __, __);`).

ZADATAK 3

Korištenjem MPI funkcija Send i Recv (skraćena sintaksa) napisati odsječak programa (proizvoljne složenosti) koji će za N procesa **ostvariti funkciju MPI_Barrier**, tj. postići da svi procesi moraju doći do istog odsječka prije nego bilo koji proces može nastaviti s izvođenjem. (U svakom procesu varijabla ID je indeks, a varijabla N ukupni broj procesa.)

ZADATAK 4

Zadan je MPI program (na slici).

Svi procesi imaju lokalne varijable a, b i c, a ID je indeks pojedinog procesa. Koje vrijednosti će imati varijabla c za svaki proces na kraju izvođenja? Navedite **sve mogućnosti** i **skicirajte redoslijed izvođenja** MPI operacija za svaki proces.

```
// Proces 1, ID = 1
MPI_Send(&ID, _, _, 2, _, _);
MPI_Recv(&a, _, _, MPI_ANY_SOURCE, _, _);
MPI_Send(&a, _, _, 3, _, _);
MPI_Recv(&b, _, _, MPI_ANY_SOURCE, _, _);
c = 2*a + b;

// Proces 2, ID = 2
MPI_Recv(&a, _, _, MPI_ANY_SOURCE, _, _);
MPI_Recv(&b, _, _, MPI_ANY_SOURCE, _, _);
c = 2*a + b;
MPI_Send(&c, _, _, 1, _, _);
MPI_Send(&ID, _, _, 3, _, _);

// Proces 3, ID = 3
MPI_Send(&ID, _, _, 2, _, _);
MPI_Recv(&a, _, _, MPI_ANY_SOURCE, _, _);
MPI_Recv(&b, _, _, MPI_ANY_SOURCE, _, _);
MPI_Send(&a, _, _, 1, _, _);
c = 2*a + b;
```

DZ1 – N FILOZOFA

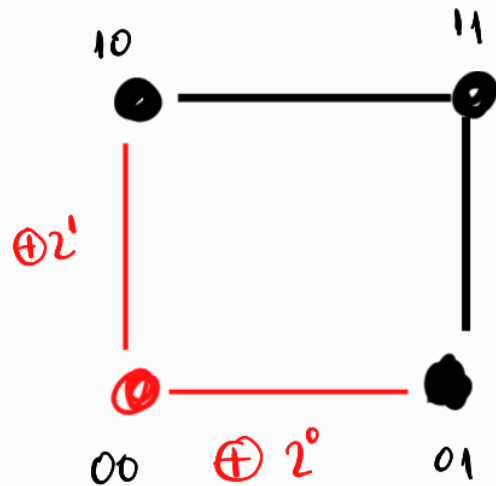
- Provjera poruka dok filozof misli?
- Čiste vs. prljave vilice?
- $n=2!$

ZADATAK 5

U MPI programu svaki proces ima lokalnu vrijednost u varijabli x . Korištenjem MPI funkcija `Send` i `Recv` (skraćena sintaksa) napisati odsječak programa **logaritamske složenosti** (po pitanju broja poslanih poruka) koji će za N procesa izračunati **minimum svih lokalnih vrijednosti**, tako **da svi procesi znaju rezultat**. (U svakom procesu varijabla ID je indeks, a varijabla N ukupni broj procesa.)

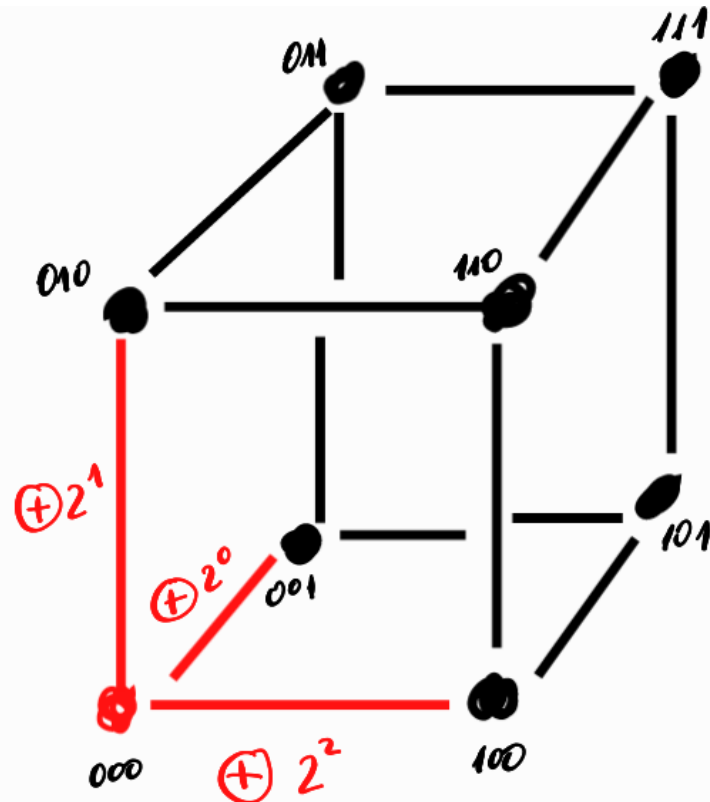
HIPERKOČKA

- $n=2$



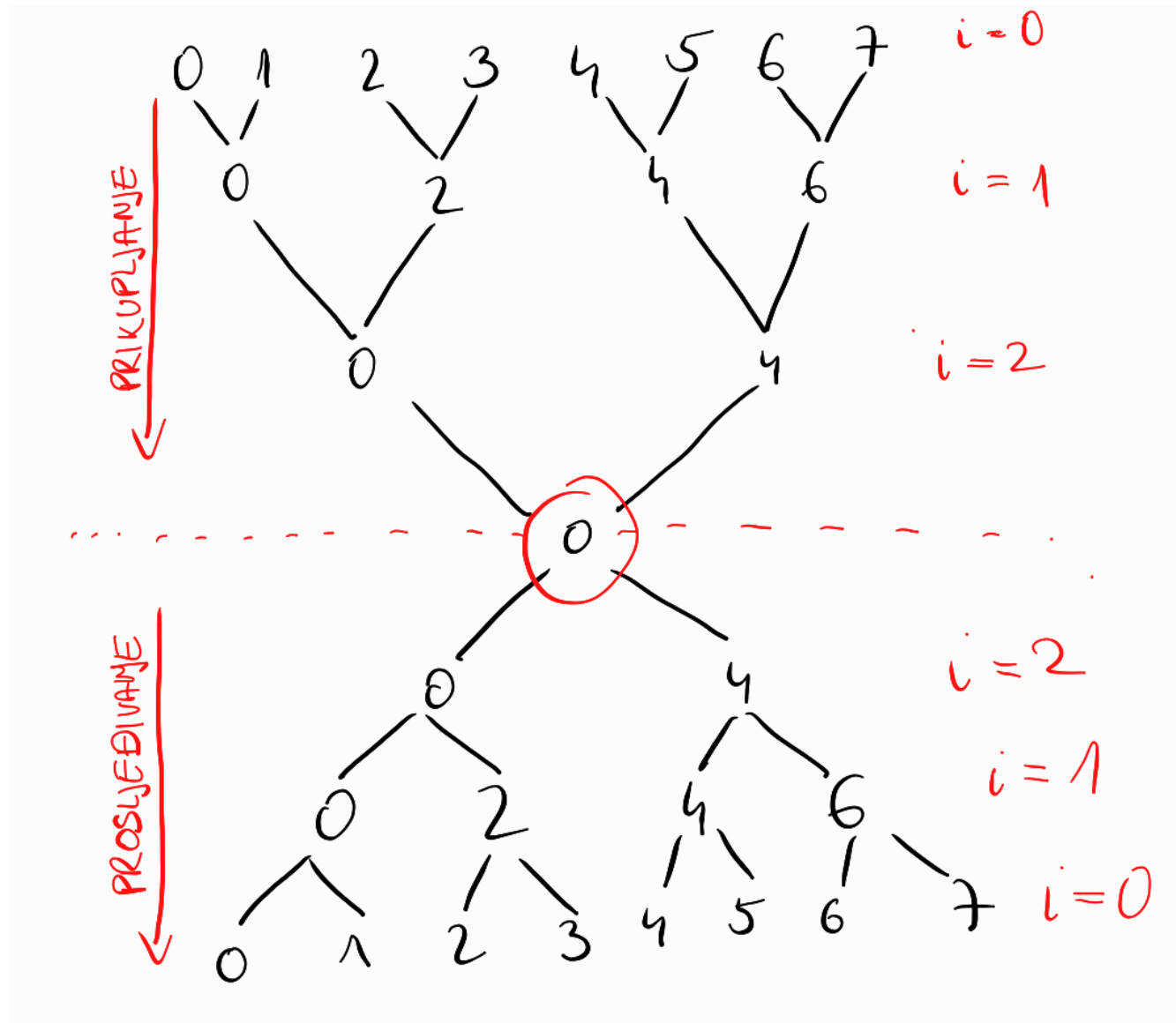
Grayev kod!

- $n=3$



[https://en.wikipedia.org/wiki/Hypercube_\(communication_pattern\)](https://en.wikipedia.org/wiki/Hypercube_(communication_pattern))

BINARNO STABLO



ZADATAK 6

U MPI programu u nekom trenutku **pojavi se promatrani događaj** unutar jednog MPI procesa. Svi procesi znaju da se događaj pojavio, ali nijedan proces (osim dotičnog, izvorišnog procesa) ne zna unutar kojeg procesa se pojavio događaj (tj. koji je proces izvorišni). Korištenjem MPI funkcija Send i Recv (skraćena sintaksa) napisati odsječak programa **logaritamske složenosti** (po pitanju broja poslanih poruka) koji će za N procesa omogućiti da **svi procesi saznaju indeks izvorišnog procesa**. (U svakom procesu varijabla ID je indeks, a varijabla N ukupni broj procesa)

ZADATAK 7

U MPI programu u nekom trenutku svih N procesa treba obaviti kritični odsječak. Svaki proces zna svoj **redni broj ulaska u K.O.**, ali ne zna redne brojeve ostalih procesa. Korištenjem MPI funkcija Send i Recv (skraćena sintaksa) napisati odsječak programa **logaritamske složenosti** (po pitanju broja poslanih poruka) koji će omogućiti da **svaki proces sazna indeks svog neposrednog prethodnika i sljedbenika** (pozivanje K.O. nije potrebno prikazati). U svakom procesu varijabla ID je indeks procesa, a varijabla RBR redni broj ulaska u K.O.