Message Passing Interface (MPI)

Συναρτήσεις επικοινωνίας κόμβου με κόμβο

Επικοινωνία κόμβου με κόμβο (Point-to-Point Communication) : επικοινωνία στην οποία εμπλέκονται μόνο δύο διεργασίες.

Συλλογική επικοινωνία (Collective Communication): επικοινωνία στην οποία εμπλέκονται περισσότερες από δύο διεργασίες.

Δύο βασικές μορφές κόμβου με κόμβο επικοινωνίας:

Αναστέλλουσα (blocking) επικοινωνία: η συνάρτηση αποστολής ή παραλαβής αναστέλλει την εκτέλεση της διεργασίας που την καλεί, έως ότου ολοκληρωθεί η διαδικασία αποστολής ή παραλαβής, αντίστοιχα.

Μη αναστέλλουσα (non-blocking) επικοινωνία: η συνάρτηση αποστολής/παραλαβής επιστρέφει αμέσως, άσχετα με το αν έχει *ολοκληρωθεί* η διαδικασία αποστολής ή παραλαβής, αντίστοιχα.

Η έννοια της *ολοκλήρωσης* της διαδικασίας αποστολής ή παραλαβής προσδιορίζεται ανάλογα με την κατάσταση επικοινωνίας (communication mode). Το MPI υποστηρίζει τέσσερις καταστάσεις επικοινωνίας.

Κάθε MPI μήνυμα αποτελείται από:

- **το φάκελο (envelope) και**
- **το κυρίως μήνυμα (message body).**

Ο φάκελος ενός MPI μηνύματος περιλαμβάνει:

- **Το όνομα της διεργασίας-αφετηρίας (source).**
- **Το όνομα της διεργασίας-προορισμού (destination).**
- **Τον communicator** στον οποίον ανήκουν οι διεργασίες.
- **Μία ετικέτα (tag)** η οποία μπορεί να χρησιμοποιηθεί προκειμένου η διεργασία-παραλήπτης να ξεχωρίσει το μήνυμα.

Έτσι, αν ληφθούν δύο μηνύματα από την ίδια διεργασία (και με ίδιες και τις υπόλοιπες παραμέτρους ταυτοποίησης), με τη χρήση της ετικέτας ο παραλήπτης μπορεί να τα διακρίνει.

Το κυρίως μήνυμα αποτελείται από :

- Τη **διεύθυνση αρχής της ενδιάμεσης μνήμης (buffer)** όπου βρίσκονται τα προς αποστολή δεδομένα ή πρόκειται να αποθηκευθούν τα προς παραλαβή δεδομένα
- Τον **τύπο των δεδομένων του μηνύματος** (datatype). Δυνατότητα χρήσης τύπων δεδομένων που ορίζονται από το χρήστη.
- Τον **αριθμό των αντικειμένων (count)** του τύπου δεδομένων που πρόκειται να αποσταλούν ή να παραληφθούν.

Βασικοί τύποι δεδομένων του MPI που αντιστοιχούν σε τύπους της C

MPI_CHAR

MPI_SHORT

MPI_INT

MPI_LONG

MPI_UNSIGNED_CHAR

MPI_UNSIGNED_SHORT

MPI_UNSIGNED

MPI_UNSIGNED_LONG

MPI_FLOAT

MPI_DOUBLE

MPI_LONG_DOUBLE

Επιπλέον τύποι δεδομένων:

- **MPI_BYTE :** παριστάνει 8 δυαδικά ψηφία. Χρησιμοποιείται όταν θέλουμε να αποστέλλουμε μηνύματα με bit-πεδία, όπου, η τιμή κάθε bit παριστάνει την τιμή ενός flag (πληροφορίες ελέγχου) στον παραλήπτη.
- **MPI_PACKED:** χρησιμοποιείται για την αποστολή και τη λήψη συνεπτυγμένων (packed) μηνυμάτων.
- **MPI_Comm:** communicator
- **MPI_Status:** δομή η οποία περιέχει πληροφορία για την κατάσταση (status) των MPI κλήσεων
- **MPI_Datatype:** δομή για την αναπαράσταση των (custom) αντικειμένων τύπου που ορίζει ο χρήστης

Η συνάρτηση MPI_Send

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

- **buf:** Η διεύθυνση αρχής της ενδιάμεσης μνήμης όπου βρίσκεται το μήνυμα.
- **count:** Ο αριθμός των προς αποστολή στοιχείων (για λόγους μεταφερσιμότητας του κώδικα σε διαφορετικές αρχιτεκτονικές, όχι το μέγεθος σε bytes).
- **datatype:** Ο τύπος των στοιχείων
- **dest:** Η τάξη της διεργασίας-παραλήπτη.
- **tag:** Η ετικέτα του μηνύματος.
- **comm:** Ο communicator στον οποίο λαμβάνει χώρα η επικοινωνία.

Παράδειγμα 1

Προκειμένου να στείλουμε έναν ακέραιο (endnum) από τη διεργασία ‘3’ στη διεργασία ‘5’, θα πρέπει η διεργασία ‘3’ να εκτελέσει τη συνάρτηση:

MPI_Send(&endnum, 1, MPI_INT, 5, tag1, MPI_COMM_WORLD);

Για να ολοκληρωθεί η επικοινωνία θα πρέπει η διεργασία ‘5’ να εκτελέσει μία αντίστοιχη συνάρτηση MPI_Recv

Η συνάρτηση MPI_Recv

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

- **buf:** Η διεύθυνση της ενδιάμεσης μνήμης όπου θα αποθηκευθεί το μήνυμα.
- **count:** Ο μέγιστος αριθμός στοιχείων προς λήψη.
- **datatype:** Ο τύπος των στοιχείων που θα παραληφθούν.
- **source:** Η τάξη της διεργασίας-αποστολέα.
- **tag:** Η ετικέτα του μηνύματος.
- **comm:** Ο communicator στον οποίο λαμβάνει χώρα η επικοινωνία.
- **status:** Μία δομή (struct) τύπου MPI_Status στην οποία επιστρέφονται πληροφορίες σχετικά με το αποτέλεσμα του receive.

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

- **status:** Η μεταβλητή της MPI_Recv από την οποία θα εξαχθεί η πληροφορία.
- **datatype:** Ο τύπος των στοιχείων που έχουν παραληφθεί.
- **count:** Επιστρέφει τον αριθμό των στοιχείων που παρελήφθησαν.

Γνωρίζοντας το μέγεθος του συγκεκριμένου datatype σε bytes και τον αριθμό των στοιχείων που ελήφθησαν, μπορούμε να υπολογίσουμε το μέγεθος του μηνύματος.

Παράδειγμα 1 (συν.)

Προκειμένου να παραληφθεί ο ακέραιος (endnum) (που εστάλη από τη διεργασία ‘3’) από τη διεργασία ‘5’, τότε η τελευταία (‘5’) θα πρέπει να εκτελέσει τη συνάρτηση:

```
MPI_Recv (&endnum, 1, MPI_INT, 3, tag1,
                        MPI_COMM_WORLD,  &status);
```

Ολοκλήρωσης κλήσης MPI_Recv: όταν τα δεδομένα του μηνύματος περιέχονται στις μεταβλητές της κλήσης της MPI_Recv.

Ολοκλήρωση κλήσης MPI_Send:

- **1η Περίπτωση: Εάν το μήνυμα αποθηκευθεί σε μία εσωτερική ενδιάμεση μνήμη του MPI για να μεταφερθεί αργότερα στον προορισμό του, η MPI_Send επιστρέφει αμέσως μετά την αντιγραφή του μηνύματος στην εσωτερική ενδιάμεση μνήμη .**
- **2η Περίπτωση: Εάν το μήνυμα πρόκειται να παραμείνει στις μεταβλητές του προγράμματος έως ότου ο παραλήπτης το παραλάβει, η MPI_Send επιστρέφει μόνον όταν *συγχρονιστούν* οι διεργασίες παραλήπτη και αποστολέα και αρχίσει η αποστολή του μηνύματος.**

- **MPI_Send και MPI_Recv: υποστηρίζουν αναστέλλουσα επικοινωνία**
- **MPI_Isend και MPI_Irecv : Υποστηρίζουν μη-αναστέλλουσα επικοινωνία**

Παράδειγμα 2: Επικοινωνία δύο διεργασιών

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

main(int argc, char **argv)
{
    char msg[20];
    int process_rank, tag  = 100;
    MPI_Status status;
```

```
 MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &process_rank);
    if (process_rank  == 0)
     {
       strcpy(msg, "Hello World");
       MPI_Send(msg, strlen(msg)+1, MPI_CHAR, 1, tag,
              MPI_COMM_WORLD);
     }
    else if (process_rank == 1)
     {
       MPI_Recv(msg, 20, MPI_CHAR, 0, tag,
                    MPI_COMM_WORLD, &status);
       printf("Process 1 Message = %s\n", msg);
     }
    MPI_Finalize();
}
```

Άσκηση: Ping-Pong (pingpong.c)

```
#include <stdio.h>
#include "mpi.h“
int main(argc,argv)
int argc;
char *argv[];
{
int numtasks, rank, dest, source, tag = 1;
char msg1[15], msg2[15];
MPI_Status stat;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
 sprintf(msg1, "Sample message");
```

Άσκηση: Ping-Pong

```
if (rank == 0)  {
  dest = 1;
  source = 1;

  while (1)  {
  MPI_Send(msg1, 15, MPI_CHAR, dest, tag,
          MPI_COMM_WORLD);
  sprintf(msg1, "\0");
  MPI_Recv(msg1, 15, MPI_CHAR, source, tag,
          MPI_COMM_WORLD, &stat);
  printf("Process  %d  Message = %s \n", rank, msg1);
  sleep(2);
  } }
```

Άσκηση: Ping-Pong

```
else if (rank == 1)  {
  dest = 0;
  source = 0;

  while (1)  {
    sprintf(msg2, "\0");
    MPI_Recv(msg2, 15, MPI_CHAR, source, tag,
          MPI_COMM_WORLD, &stat);
    printf("Process %d Message = %s \n", rank, msg2);
    sleep(2);
    MPI_Send(msg2, 15, MPI_CHAR, dest, tag,
          MPI_COMM_WORLD);
  } }
 MPI_Finalize();
}
```

Άσκηση : Ping-Pong with a Limit (ping_pong_limit.c)

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char** argv) {
const int LIMIT = 12;
int rank;
int size;
MPI_Init(&argc, &argv);
MPI_Status stat;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
 int mes;
 // We need  only 2 processes
 if (size != 2) {
  fprintf(stderr, "The number of processes must be two for %s\n", argv[0]);
  MPI_Abort(MPI_COMM_WORLD, 1);
 }
```

Άσκηση: Ping-Pong with Limit (ping_pong_limit.c)

```
int count = 0;
  int partner_rank = (rank + 1) % 2;
  while (count < LIMIT) {
    if (rank == count % 2) {
          count++;
      MPI_Send(&count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD);
      printf("Process %d sent  message %d to process %d\n",
           rank, count, partner_rank);
    } else {
      MPI_Recv(&count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD,
            &stat);
      printf("Process %d received message  %d from %d\n",
           rank, count, partner_rank);
    }
  }
  MPI_Finalize();
}
```

Άσκηση: Διακίνηση μηνύματος σε δακτύλιο (ring1.c)

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char** argv) {

int rank;
int size;

MPI_Init(&argc, &argv);
MPI_Status stat;

MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

int mes;
```

Άσκηση: (ring1.c)

```
if (rank != 0) {
   MPI_Recv(&mes, 1, MPI_INT, rank - 1, 0, MPI_COMM_WORLD,
            &stat);
   printf("Process %d received message %d from process %d\n", rank, mes,
         rank - 1);
   } else   {
     // if you are process 0
     mes = 100;
  }
  MPI_Send(&mes, 1, MPI_INT, (rank + 1) % size, 0, MPI_COMM_WORLD);
   if (rank == 0) {
   MPI_Recv(&mes, 1, MPI_INT, size - 1, 0, MPI_COMM_WORLD, &stat);
   printf("Process %d received message %d from process %d\n", rank, mes,
         size - 1);
  }
  MPI_Finalize();  }
```

Άσκηση: Υπολογισμός $S = 1^2 + 2^2 + ... + n^2$ (nsquare.c)

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

main(int argc, char** argv) {
int my_rank;
int p,k,res,finres,a1,b1,num;
int source;
int target;
int tag1 = 50;
int tag2 = 60;
int plithos;
MPI_Status status;

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);
```

```
if (my_rank == 0)  {
   printf("Dose plithos arithmon:");
   scanf("%d", &plithos);
   for (target = 1; target < p; target++)
         MPI_Send(&plithos, 1, MPI_INT, target, tag1,
                  MPI_COMM_WORLD);
 }
else
   MPI_Recv(&plithos, 1, MPI_INT, 0, tag1, MPI_COMM_WORLD,
          &status);
   res = 0;
   num = plithos/p;
   a1 = (my_rank * num) + 1;
   b1 = a1 + num - 1;

   for (k=a1; k<=b1; k++)
       res = res + (k*k);
```

```
if (my_rank != 0)  {
   MPI_Send(&res, 1, MPI_INT, 0, tag2, MPI_COMM_WORLD); }
 else
 {  finres = res;
   printf("\n Apotelesma of process %d: %d\n",
          my_rank, res);

   for (source = 1; source < p; source++)  {
      MPI_Recv(&res, 1, MPI_INT, source, tag2,
             MPI_COMM_WORLD, &status);
       finres = finres + res;
      printf("\n Apotelesma of process %d: %d\n",
            source, res);
    }
 printf("\n\n\n Teliko Apotelesma: %d\n", finres);
}
 MPI_Finalize();
}
```

Άσκηση: Υπολογισμός $S = a_1^2 + a_2^2 + ... + a_n^2$ (squares.c)

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

main(int argc, char** argv)
{
  int my_rank;
  int p,k,res,finres,num;
  int source,target;
  int tag1=50, tag2=60, tag3=70;
  int plithos;
  int data[100];
  int data_loc[100];
  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
  MPI_Comm_size(MPI_COMM_WORLD, &p);
```

```
if (my_rank == 0)  {
      printf("Dose plithos aritmon:\n");
      scanf("%d", &plithos);
      printf("Dose tous %d arithmous:\n", plithos);
      for (k=0; k<plithos; k++)
          scanf("%d", &data[k]);
      for (target = 1; target < p; target++)
          MPI_Send(&plithos, 1, MPI_INT, target, tag1, MPI_COMM_WORLD);
     num = plithos/p;  k=num;
     for (target = 1; target < p; target++)  {
       MPI_Send(&data[k], num, MPI_INT, target, tag2, MPI_COMM_WORLD);
       k+=num;  }
     for (k=0; k<num; k++)
       data_loc[k]=data[k];
    }
  else  {
      MPI_Recv(&plithos, 1, MPI_INT, 0, tag1, MPI_COMM_WORLD, &status);
      num = plithos/p;
      MPI_Recv(&data_loc[0], num, MPI_INT, 0, tag2, MPI_COMM_WORLD,
&status);
    }
```

```
res = 0;
 for (k=0; k<num; k++)
     res = res + (data_loc[k]*data_loc[k]);
 if (my_rank != 0)  {
    MPI_Send(&res, 1, MPI_INT, 0, tag3, MPI_COMM_WORLD);
  }
 else  {
    finres = res;
    printf("\n Apotelesma of process %d: %d\n", my_rank, res);
    for (source = 1; source < p; source++)  {
       MPI_Recv(&res, 1, MPI_INT, source, tag3, MPI_COMM_WORLD, &status);
        finres = finres + res;
        printf("\n Apotelesma of process %d: %d\n", source, res);
     }
  printf("\n\n\n Teliko Apotelesma: %d\n", finres);
 }
 MPI_Finalize();          }
```

Άσκηση: Παράλληλη Αναζήτηση (search.c)

```
#include <stdio.h>
#include "mpi.h“
main(int argc, char** argv)
{
int my_rank, size;
int source, dest;
int tag1= 50;
int tag2 = 60;
int tag3 =70;
int found = 0;
int other_found;
int k, code, namelen;
int data[10];
MPI_Status status;
char proc_name[MPI_MAX_PROCESSOR_NAME];
char other_proc_name[MPI_MAX_PROCESSOR_NAME];
```

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Get_processor_name(proc_name, &namelen);

for (k=0; k<10; k++)
data[k]=my_rank+k;
if (my_rank == 0)
 {
  printf("Dose ena kodiko anazitisis:\n");
  scanf("%d", &code);
  for (dest =1; dest<size; dest++)
     MPI_Send(&code, 1, MPI_INT, dest, tag1, MPI_COMM_WORLD); }
else
 {
  MPI_Recv(&code, 1, MPI_INT, 0, tag1, MPI_COMM_WORLD, &status); }
```

```
for (k=0; k<10; k++)
   { if (data[k] == code) found=1; }

if (my_rank != 0)
   {  MPI_Send(&found, 1, MPI_INT, 0, tag2, MPI_COMM_WORLD);
      MPI_Send(proc_name, namelen+1, MPI_CHAR, 0, tag3, MPI_COMM_WORLD);  }
if (my_rank == 0)  {
 for (source=1; source<size; source++)  {
    MPI_Recv(&other_found, 1, MPI_INT, source, tag2, MPI_COMM_WORLD, &status);
    MPI_Recv(other_proc_name, 50, MPI_CHAR, source, tag3, MPI_COMM_WORLD,
     &status);
     if (other_found)
        printf("\n Code %d found in database of process %d on %s \n", code, source,
     other_proc_name);
     }
 if (found)
     printf("\n Code %d found in process %d on %s \n", code, my_rank, proc_name);
  }
MPI_Finalize();   }
```