|  |  |
| --- | --- |
|  | Πανεπιστήμιο Πελοποννήσου  Σχολή Μηχανικών  Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών (ΗΜΜΥ) |

Εισαγωγή στα Κατανεμημένα Συστήματα

(ECE\_Κ-650)

Εργαστηριακή Άσκηση 3:

Κυματικά Πρωτόκολλα – Πρωτόκολλα

Διάσχισης

**Ομάδα:**

**Χανκο Τζεμαι, 21150**

**Β. Ταμπακάς**

**Καθηγητής**

[tampakas@uop.gr](mailto:tampakas@uop.gr)

Εργαστηριακό Τεύχος

Εργαστηριακό Τεύχος

Α. Το πρωτόκολλο Echo

A screenshot of a computer program

Description automatically generated

Β. Το πρωτόκολλο Tarry

A screenshot of a computer program

Description automatically generated

Εργασία

1)

Κώδικας:

#include "mpi.h"

#include <stdio.h>

#include <stdlib.h>

int main( int argc, char \*argv[] )

{

int i, k, neighbourNumber,token;

int wsize;

int father;

int \*index, \*edges,\*neighbours;

MPI\_Comm comm1;

MPI\_Request request[8];

MPI\_Status status, status2;

MPI\_Init( &argc, &argv ); //preparation of environment of MPI

MPI\_Comm\_size( MPI\_COMM\_WORLD, &wsize ); // Get the number of Processors

int rank;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

if (wsize >= 3)

{

index = (int\*)malloc(wsize \* sizeof(int) );

edges = (int\*)malloc(wsize \* 2 \* sizeof(int) );

neighbours = (int\*)malloc(wsize \* 10 \* sizeof(int) );

if (!index || !edges) {

printf( "Unable to allocate %d words for index or edges\n", 3 \* wsize );

fflush(stdout);

MPI\_Abort( MPI\_COMM\_WORLD, 1 );

}

index[0]=3;

index[1]=6;

index[2]=9;

index[3]=13;

index[4]=15;

index[5]=19;

index[6]=25;

index[7]=28;

edges[0]=1;

edges[1]=6;

edges[2]=5;

edges[3]=0;

edges[4]=6;

edges[5]=7;

edges[6]=3;

edges[7]=6;

edges[8]=7;

edges[9]=2;

edges[10]=4;

edges[11]=5;

edges[12]=6;

edges[13]=3;

edges[14]=5;

edges[15]=0;

edges[16]=3;

edges[17]=4;

edges[18]=6;

edges[19]=0;

edges[20]=1;

edges[21]=2;

edges[22]=3;

edges[23]=5;

edges[24]=7;

edges[25]=1;

edges[26]=2;

edges[27]=6;

MPI\_Graph\_create( MPI\_COMM\_WORLD, wsize, index, edges, 0, &comm1 );

if (rank == 6) //Initator

{

MPI\_Graph\_neighbors\_count(comm1,rank,&neighbourNumber);

MPI\_Graph\_neighbors(comm1,rank,neighbourNumber,neighbours);

printf( "My node no is = %d and I have %d neighbours\n", rank,

neighbourNumber);

printf( "My neıghbours are: ");

for(i=0;i<neighbourNumber;i++)

{

printf("%d,",neighbours[i]);

}

printf( "\n-------------------------------------\n");

for(k=0; k<neighbourNumber; k++)

MPI\_Send(&token, 1, MPI\_INT, neighbours[k], 0, MPI\_COMM\_WORLD);

for(k=0; k<neighbourNumber; k++)

MPI\_Irecv(&token, 1, MPI\_INT, neighbours[k], 0, MPI\_COMM\_WORLD,

&request[k]);

for(k=0; k<neighbourNumber; k++)

MPI\_Wait(&request[k], &status);

printf("ECHO finised\n");

}

else

{

MPI\_Recv(&token, 1, MPI\_INT, MPI\_ANY\_SOURCE, 0, MPI\_COMM\_WORLD,

&status2);

father = status2.MPI\_SOURCE;

MPI\_Graph\_neighbors\_count(comm1,rank,&neighbourNumber);

MPI\_Graph\_neighbors(comm1,rank,neighbourNumber,neighbours);

for(k=0; k<neighbourNumber; k++)

if (neighbours[k] != father)

MPI\_Send(&token, 1, MPI\_INT, neighbours[k], 0,

MPI\_COMM\_WORLD);

for(k=0; k<neighbourNumber; k++)

if (neighbours[k] != father)

MPI\_Irecv(&token, 1, MPI\_INT, neighbours[k], 0,

MPI\_COMM\_WORLD, &request[k]);

for(k=0; k<neighbourNumber; k++)

if (neighbours[k] != father)

MPI\_Wait(&request[k], &status);

MPI\_Send(&token, 1, MPI\_INT, father, 0, MPI\_COMM\_WORLD);

printf("Node %d has node %d as father\n", rank, father);

}

MPI\_Comm\_free( &comm1 );

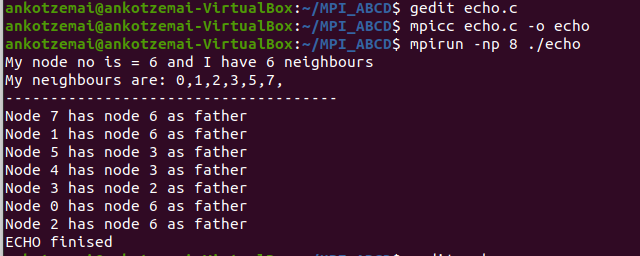
}

MPI\_Finalize(); //Finish MPI

return 0;

}

Αποτέλεσμα:



2)

Κώδικας

#include "mpi.h"

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include <limits.h>

int main( int argc, char \*argv[] )

{

int end = 0, i, neighbourNumber,token, rn, next, notFinished = 1;

int wsize = 7;

int father;

int \*index, \*edges,\*neighbours, \*visited;

int nnodes, nedges;

MPI\_Comm comm1, comm2;

MPI\_Request request[5];

MPI\_Status status, status2;

MPI\_Init( &argc, &argv ); //prepara�on of environment of MPI

MPI\_Comm\_size( MPI\_COMM\_WORLD, &wsize ); // Get the number of Processors

int rank;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

srand(time(NULL) + rank);

if (wsize >= 3) { // If Processor number is more than 3 we can make a graph.

index = (int\*)malloc(wsize \* sizeof(int) );

edges = (int\*)malloc(wsize \* 2 \* sizeof(int) );

neighbours = (int\*)malloc(wsize \* 10 \* sizeof(int) );

visited = (int\*)malloc(wsize \* 10 \* sizeof(int) );

// allocate memory for arrays

if (!index || !edges) {

printf( "Unable to allocate %d words for index or edges\n", 3 \* wsize ); //Error

//Control if we cannot allocate memory

fflush(stdout);

MPI\_Abort( MPI\_COMM\_WORLD, 1 );

}

index[0]=3;

index[1]=6;

index[2]=9;

index[3]=11;

index[4]=15;

index[5]=18;

index[6]=22;

edges[0]=1;

edges[1]=5;

edges[2]=6;

edges[3]=0;

edges[4]=2;

edges[5]=6;

edges[6]=1;

edges[7]=3;

edges[8]=4;

edges[9]=2;

edges[10]=4;

edges[11]=2;

edges[12]=3;

edges[13]=5;

edges[14]=6;

edges[15]=0;

edges[16]=4;

edges[17]=6;

edges[18]=0;

edges[19]=1;

edges[20]=4;

edges[21]=5;

MPI\_Graph\_create( MPI\_COMM\_WORLD, wsize, index, edges, 0, &comm1 );

if (rank == 6) //Initiator

{

MPI\_Graph\_neighbors\_count(comm1,rank,&neighbourNumber);

MPI\_Graph\_neighbors(comm1,rank,neighbourNumber,neighbours);

for(i=0;i<neighbourNumber;i++)

{

visited[i] = 0;

}

printf( "\n-------------TARRY STARTED------------------------\n");

//Send tokens to random neighbour

rn = rand()%neighbourNumber;

MPI\_Send(&token, 1, MPI\_INT, neighbours[rn], 0, MPI\_COMM\_WORLD);

printf("Node %d will send to %d the token\n", rank, neighbours[rn]);

fflush(stdout);

visited[rn] = 1;

while (end == 0)

{

MPI\_Recv(&token, 1, MPI\_INT, MPI\_ANY\_SOURCE, 0,

MPI\_COMM\_WORLD, &status2);

next = 0;

notFinished = 1;

while ( (next < neighbourNumber) && notFinished == 1 )

{

if ( (visited[next] == 0) && neighbours[next] != father)

notFinished = 0;

else

next++;

}

if (notFinished == 1)//all visited except father

{

end = 1;

}

else

{

printf("Node %d will send to %d the token\n", rank,

neighbours[next]);

fflush(stdout);

sleep(1);

MPI\_Send(&token, 1, MPI\_INT, neighbours[next], 0,

MPI\_COMM\_WORLD);

visited[next] = 1;

}

}

printf( "\n-------------TARRY FINISHED------------------------\n");

}

else

{

MPI\_Recv(&token, 1, MPI\_INT, MPI\_ANY\_SOURCE, 0, MPI\_COMM\_WORLD,

&status2);

father = status2.MPI\_SOURCE;

MPI\_Graph\_neighbors\_count(comm1,rank,&neighbourNumber);

MPI\_Graph\_neighbors(comm1,rank,neighbourNumber,neighbours);

for(i=0;i<neighbourNumber;i++)

{

visited[i] = 0;

}

while (end == 0)

{

next = 0;

notFinished = 1;

while ( (next < neighbourNumber) && notFinished == 1 )

{

if ( (visited[next] == 0) && neighbours[next] != father)

notFinished = 0;

else

next++;

}

if (notFinished == 1)//all visited except father

{

printf("Node %d will send to father %d the token\n", rank,

father);

fflush(stdout);

sleep(1);

MPI\_Send(&token, 1, MPI\_INT, father, 0, MPI\_COMM\_WORLD);

end = 1;

}

else

{

printf("Node %d will send to %d the token\n", rank,

neighbours[next]);

fflush(stdout);

sleep(1);

MPI\_Send(&token, 1, MPI\_INT, neighbours[next], 0,

MPI\_COMM\_WORLD);

visited[next] = 1;

MPI\_Recv(&token, 1, MPI\_INT, MPI\_ANY\_SOURCE, 0,

MPI\_COMM\_WORLD, &status2);

}

}

}

MPI\_Comm\_free( &comm1 ); //Empty comm1 and give to system.

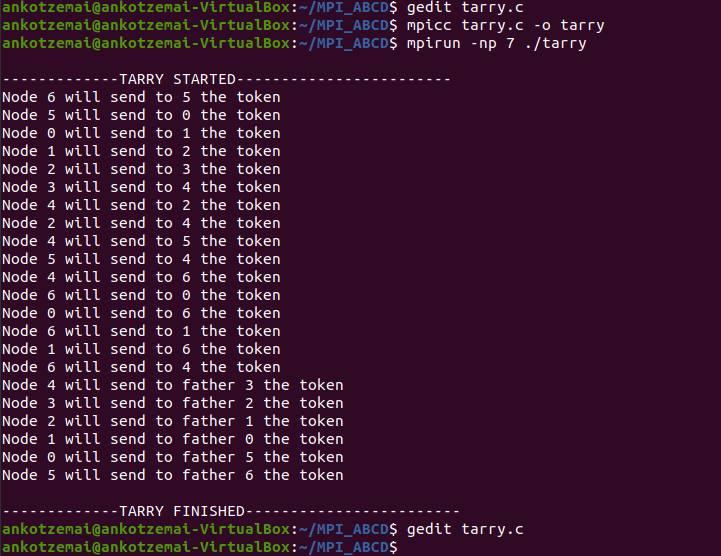
}

MPI\_Finalize(); //Finish MPI

return 0;

}

Αποτέλεσμά



3)

Κώδικας

#include "mpi.h"

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include <limits.h>

int main( int argc, char \*argv[] )

{

int end = 0, i, neighbourNumber,token, rn, next, notFinished = 1;

int wsize = 7;

int father;

int \*index, \*edges,\*neighbours, \*visited;

int \*status;

int nnodes, nedges;

MPI\_Comm comm1, comm2;

MPI\_Request request[5];

MPI\_Status status, status2;

MPI\_Init( &argc, &argv ); //prepara�on of environment of MPI

MPI\_Comm\_size( MPI\_COMM\_WORLD, &wsize ); // Get the number of Processors

int rank;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

srand(time(NULL) + rank);

if (wsize >= 3) { // If Processor number is more than 3 we can make a graph.

status = (inr\*)malloc(wsize\* sizeof(int));

index = (int\*)malloc(wsize \* sizeof(int) );

edges = (int\*)malloc(wsize \* 2 \* sizeof(int) );

neighbours = (int\*)malloc(wsize \* 10 \* sizeof(int) );

visited = (int\*)malloc(wsize \* 10 \* sizeof(int) );

// allocate memory for arrays

if (!index || !edges) {

printf( "Unable to allocate %d words for index or edges\n", 3 \* wsize ); //Error

//Control if we cannot allocate memory

fflush(stdout);

MPI\_Abort( MPI\_COMM\_WORLD, 1 );

}

srand(0);

index[0]=5;

index[1]=9;

index[2]=12;

index[3]=15;

index[4]=19;

index[5]=23;

index[6]=26;

index[7]=28;

edges[0]=1;

edges[1]=4;

edges[2]=5;

edges[3]=6;

edges[4]=7;

edges[5]=0;

edges[6]=2;

edges[7]=3;

edges[8]=4;

edges[9]=1;

edges[10]=3;

edges[11]=4;

edges[12]=1;

edges[13]=2;

edges[14]=5;

edges[15]=0;

edges[16]=1;

edges[17]=2;

edges[18]=5;

edges[19]=0;

edges[20]=3;

edges[21]=4;

edges[22]=6;

edges[23]=0;

edges[24]=5;

edges[25]=7;

edges[26]=0;

edges[27]=6;

MPI\_Graph\_create( MPI\_COMM\_WORLD, wsize, index, edges, 0, &comm1 );

if (rank == 6) //Initiator

{

MPI\_Graph\_neighbors\_count(comm1,rank,&neighbourNumber);

MPI\_Graph\_neighbors(comm1,rank,neighbourNumber,neighbours);

for(i=0;i<neighbourNumber;i++)

{

visited[i] = 0;

status[i] = rand() % 2;

}

printf( "\n-------------TARRY STARTED------------------------\n");

//Send tokens to random neighbour

rn = rand()%neighbourNumber;

MPI\_Send(&token, 1, MPI\_INT, neighbours[rn], 0, MPI\_COMM\_WORLD);

printf("Node %d will send to %d the token\n", rank, neighbours[rn]);

fflush(stdout);

visited[rn] = 1;

while (end == 0)

{

MPI\_Recv(&token, 1, MPI\_INT, MPI\_ANY\_SOURCE, 0,

MPI\_COMM\_WORLD, &status2);

next = 0;

notFinished = 1;

while ( (next < neighbourNumber) && notFinished == 1 )

{

if ( (visited[next] == 0) && neighbours[next] != father)

notFinished = 0;

else

next++;

}

if (notFinished == 1)//all visited except father

{

end = 1;

}

else

{

printf("Node %d will send to %d the token\n", rank,

neighbours[next]);

fflush(stdout);

sleep(1);

MPI\_Send(&token, 1, MPI\_INT, neighbours[next], 0,

MPI\_COMM\_WORLD);

visited[next] = 1;

}

}

printf( "\n-------------TARRY FINISHED------------------------\n");

}

else

{

MPI\_Recv(&token, 1, MPI\_INT, MPI\_ANY\_SOURCE, 0, MPI\_COMM\_WORLD,

&status2);

father = status2.MPI\_SOURCE;

MPI\_Graph\_neighbors\_count(comm1,rank,&neighbourNumber);

MPI\_Graph\_neighbors(comm1,rank,neighbourNumber,neighbours);

for(i=0;i<neighbourNumber;i++)

{

visited[i] = 0;

}

while (end == 0)

{

next = 0;

notFinished = 1;

while ( (next < neighbourNumber) && notFinished == 1 )

{

if ( (visited[next] == 0) && neighbours[next] != father)

notFinished = 0;

else

next++;

}

if (notFinished == 1)//all visited except father

{

printf("Node %d will send to father %d the token\n", rank,

father);

fflush(stdout);

sleep(1);

MPI\_Send(&token, 1, MPI\_INT, father, 0, MPI\_COMM\_WORLD);

end = 1;

}

else

{

printf("Node %d will send to %d the token\n", rank,

neighbours[next]);

fflush(stdout);

sleep(1);

MPI\_Send(&token, 1, MPI\_INT, neighbours[next], 0,

MPI\_COMM\_WORLD);

visited[next] = 1;

MPI\_Recv(&token, 1, MPI\_INT, MPI\_ANY\_SOURCE, 0,

MPI\_COMM\_WORLD, &status2);

}

}

}

MPI\_Comm\_free( &comm1 ); //Empty comm1 and give to system.

}

MPI\_Finalize(); //Finish MPI

return 0;

}

Αποτέλεσμά

A screenshot of a computer

Description automatically generated