ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

UNIVERSITY OF WEST ATTICA

ΛΟΓΟΘΕΤΗΣ ΑΛΕΞΙΟΣ-ΑΝΤΩΝΙΟΣ

ICE 20390132

ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΠΑΡΑΛΛΗΛΟ ΥΠΟΛΟΓΙΣΜΟ

ΕΡΓΑΣΙΑ ΠΡΩΤΗ ΕΡΓΑΣΤΗΡΙΑΚΟ ΤΜΗΜΑ Ε1

06/12/2022

Αρχικά ξεκινάμε με την δήλωση μεταβλητών , έπειτα δηλώνουμε στο πρόγραμμα οτι απο εδώ και κάτω θα υπάρχουν εντολές του MPI με την εντολή MPI_Iinit(&argc,&argv).Μετα με τις εντολές MPI_Comm_rank και MPI_Comm size δίνουμε στις μεταβλητες my_rank και p τον αριθμό της κάθε διεργασίας και το πλήθος των διεργασιών που δημιουργούνται, εδω πρέπει να τονιστεί οτι κάθε διεργασία δημιουργεί εινα δικό της instance της κάθε μεταβλητής στην δική της μνήμη.

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

int main(int argc, char **argv)
{
    int my_rank;
    int tmpflag1,tmp;
    int p, k, flag, finflag, num;
    int source, target;
    int tag1 = 10, tag2 = 20, tag3 = 30;
    int plithos;
    int data[100];
    int data_loc[100];
    int pos;
    int tmppos;
    int i;
    int ch;
    MPI_Status status;

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

Εδώ αρχικά μπαίνουμε σε μια do while η οποία χρησιμοποιείται για την επναληπτική διαδικασία του μενού που θα δούμε αργότερα.Έπειτα αρχικοποιούμε μερικές μεταβλητές οι οποίες επίσης θα χρησιμοποιηθούν αργότερα.Μετά η διεργασία μηδέν θα ζητήσει απο τον χρήστη το πλήθος των αριθμών ,αν αυτο είναι μικρότερο απο τον αριθμό των διεργασιών τότε όπως θα φανεί και παρακάτω τυπώνουμε ένα σχετικό μήνυμα και τερματίζουμε την εκτέλεση του προγράμματος.Έαν το πλήθος των αριθμών δεν είναι μικρότερο όμως ζητάμε απο τον χρήστη τους αριθμούς αυτους και τους περνάμε σε έναν πίνακα , μετά στέλνουμε το πλήθος αυτο στις υπόλοιπες διεργασίες και υπολογίζουμε το πλήθος των αριθμών που θα πάρει καθε διεργασια αποθηκέυοντάς το στο num.

```
do{
    tmpflag1=flag=finflag=0; /*Flags for checking if the sequence is sorted*
    pos=tmppos=-1;
      if (my_rank == 0){/*this part of the code will only be run by the 0th
        system("clear");
        printf("The amount of numbers must be bigger than :: %d !\n",p); /*p
        printf("Input the amount of numbers : ");
        scanf("%d", &plithos);
        if(plithos>=p){ /*if the amount of numbers is smaller than the amoun
            printf("Input the %d numbers : \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;  /*spliting the table to equal pieces*/
```

Εδώ αποθηκέυουμε στον “τοπικό” πίνακα της διεργασίας 0 τα πρώτα num στοιχεία. Έπειτα στέλνουμε και στις υπόλοιπες διεργασίες τα επόμενα num στοιχεία εκτός απο την τελευταία διεργασία η οποία στην περίπτωση που το πλήθος των αριθμών δεν είναι πολλαπλάσιο του πλήθους των διεργασιών τοτε θα πάρει τα υπόλοιπα num + (Υπόλοιπο(plithos/p)).

```
for (k = 0; k < num; k++)
    data_loc[k] = data[k];  /*assigning the first num numbers to data

for (target = 1; target < p; target++){
    if(target==p-1){
        if(plithos%p!=0){
            num=num+plithos%p;  /* checks if  (amount of numbers)%(ar
        }                       /* if not then add the remainder done

    }
    MPI_Send(&data[k], num, MPI_INT, target, tag2,MPI_COMM_WORLD); /*
    k += num;                                                      /*
}

num = plithos/p; /*calculating num again in case the if(plithos%p!=0)
```

Παρακάτω βλέπουμε τον τερματισμό του προγράμματος που αναφέρθηκε νωρίτερα.

```
    }else{  /*p0 exiting here*/
        printf("The amount of numbers must be bigger than the amount of threads!\n");
        for(target=1;target<p;target++)
            MPI_Send(&plithos, 1, MPI_INT, target, tag1,MPI_COMM_WORLD);
        printf("Thread %d is exiting!\n",my_rank);
        MPI_Finalize();
        exit(1);
    }
}else{
    /*rest of processes exit here*/
    MPI_Recv(&plithos, 1, MPI_INT, 0, tag1, MPI_COMM_WORLD, &status);
    if(plithos<p){
        printf("Thread %d is exiting!\n",my_rank);
        MPI_Finalize();
        exit(1);
    }
```

Έπειτα βλέπουμε την παραλαβή των δεδομένων αυτών απο ολες τις διεργασίες, εκτός της διεργασίας 0, στους “τοπικούς” πινακές data_loc.

```
    num = plithos / p;
    if(my_rank==p-1){
        if(plithos%p!=0){
            num=num+(plithos%p);
            // printf(":: %d %%\n",num);
        }
    }
    MPI_Recv(&data_loc[0], num, MPI_INT, 0, tag2, MPI_COMM_WORLD,&status);


}
```

Στο παρακάτω στιγμιότυπο βλέπουμε τον τρόπο με τον οποίο θα γίνει ο έλεγχος για το εάν είναι ταξινομημένος ο πίνακας.Αρχικά κάθε διεργασία θα κοιτάξει για τους δικούς της num αριθμούς αν υπάρχει κάποιο λάθος στην ταξινόμηση , έαν υπάρχει θα αλλάξουμε την τιμή του flag και θα υπολογίσουμε την θέση του λάθους πολλαπλασιάζοντας τον αριθμό της διεργασίας με το πλήθος των αριθμών που ισομοιράστηκε σε κάθε διεργασία και προσθέτοντας τον αριθμό των επαναλήψεων . Για να γίνει πιο κατανοητό θα δείξω και ένα παράδειγμα. Έστω οτι έχουμε 3 διεργασίες και το πλήθος των αριθμών είναι το 6. Αυτό σημαίνει οτι κάθε διεργασία θα λάβει απο δύο αριθμούς. Έστω λοιπόν η αριθμοί 1 2 | 4 3 | 5 6 . Η κάθετος συμβολίζει το μοίρασμα στις διεργασίες . Τρεχοντας λοιπόν τον κώδικα η δεύτερη διεργασία ,δηλαδη η διεργασία 1 εφόσον ξεκινάει απο το 0, βρίσκει στην πρώτη της επανάληψη οτι υπάρχει λάθος αρα η πραξη που γίνεται είναι ίση με :

pos= k + (plithos/p)*my_rank => pos=0 +(6/3)*1 => pos = 2*1 = 2.

Αν ξεκινήσουμε να μετράμε απο το 0 θα δούμε οτι πράγματι η τρίτη σύγκριση ,(4 > 3) είναι αληθής άρα η αριθμοί μας δέν είναι ταξινομημένη . Ο λόγος που δεν χρησιμοποιούμε το num και χρησιμοιποιούμε το plithos/p είναι διότι σε περίπτωση που το plithos%p !=0 τότε η τελευταία διεργασία θα υπολογίσει μεγαλύτερο αριθμο αφου θα έχει προσθέσει στο num της και το υπόλοιπο .

```
    /*checking if its sorted if not check the flag*/
    for (k = 0; k < num - 1; k++){
        if (data_loc[k] > data_loc[k + 1]){
            flag = 1;
            pos=k + (plithos/p)*my_rank; /*calculating the position */
        }

    }
```

Στο επόμενο στιγμιότυπο βλέπουμε την σύγκριση που πρέπει να γίνει στους ενδιάμεσους αριθμούς δηλαδή σύμφωνα με το προηγούμενο παράδειγμα

1 2 | 4 3 | 5 6 , τις συγκρίσεις 2>4 και 5>6.Οι αριθμοί αυτοί βρίσκονται σε ξεχωρίστες διεργασίες οπότε θα πρέπει να στείλουν τα δεδομένα ο ένας στον αλλον και να κάνουμε τον ίδιο έλεγχο .

```
if(my_rank!=p-1){   /*if you're not the last process then send your last numb
    MPI_Send(&data_loc[k], 1, MPI_INT, my_rank+1, tag3, MPI_COMM_WORLD);
    //   printf("my rank %d :: sending  %d \n",my_rank,data_loc[k]);
    i=k + (plithos/p)*my_rank;
    MPI_Send(&i, 1, MPI_INT, my_rank+1, tag3, MPI_COMM_WORLD);
}


if(my_rank!=0){   /*recieving the last number from previous process and compa
    MPI_Recv(&tmp, 1, MPI_INT,my_rank-1, tag3, MPI_COMM_WORLD, &status);
    MPI_Recv(&tmppos, 1, MPI_INT,my_rank-1, tag3, MPI_COMM_WORLD, &status);
    if(tmp>data_loc[0]){
        flag=1;
        pos=tmppos;
    }

}
```

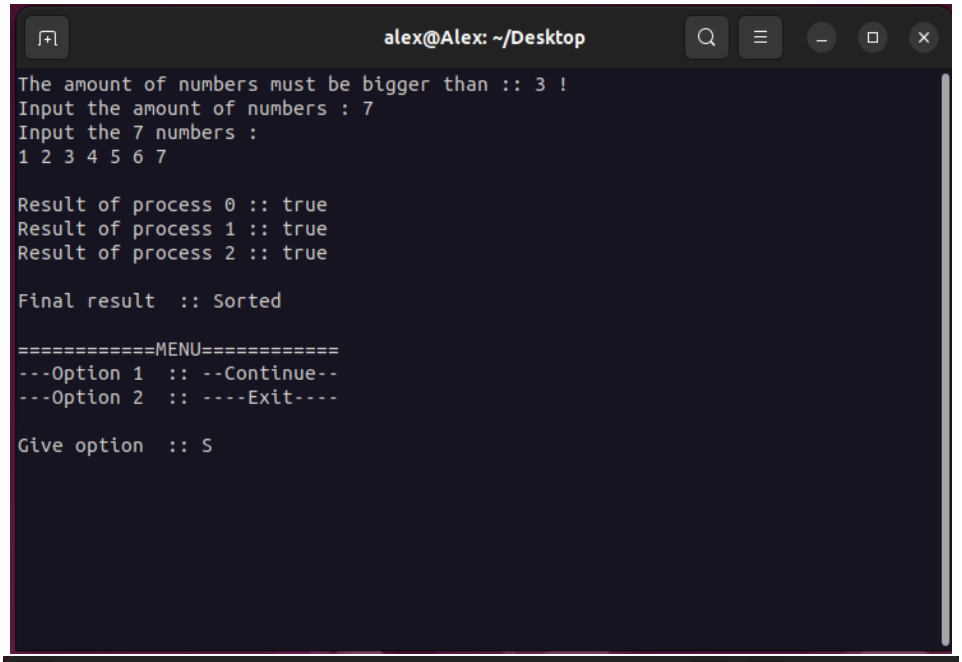
```
if (my_rank != 0){  /*sending your results to p0*/
    MPI_Send(&flag, 1, MPI_INT, 0, tag1, MPI_COMM_WORLD);
    MPI_Send(&pos, 1, MPI_INT, 0, tag1, MPI_COMM_WORLD);
}else{
        /*printing result of p0*/
    printf("\nResult of process %d :: ", my_rank);
    if(flag==0)printf("true\n");else{ printf("false\n");finflag=1;}

    for (source = 1; source < p; source++){
        MPI_Recv(&tmpflag1, 1, MPI_INT, source, tag1, MPI_COMM_WORLD, &status);
        MPI_Recv(&tmppos, 1, MPI_INT, source, tag1, MPI_COMM_WORLD, &status);
        if (tmpflag1 == 1)
            finflag = 1;

        /*if pos hasent been changed and process[source] found a mistake in the sequence*/
        /*then change the pos to tmppos so now it cannot change again thus making sure  */
        /*that we print the first mistake that has been found*/
        if(pos==-1 && tmppos!=-1)
            pos=tmppos;

        printf("Result of process %d :: ", source);
        if(tmpflag1==0)printf("true\n");else printf("false\n");
    }

    printf("\nFinal result  :: ");
    if(finflag==0)
        printf("Sorted\n");
    else
        printf("Not sorted\n\nSequence breaks first at pos %d , %d > %d \n",pos,data[pos],data[pos+1]);
```

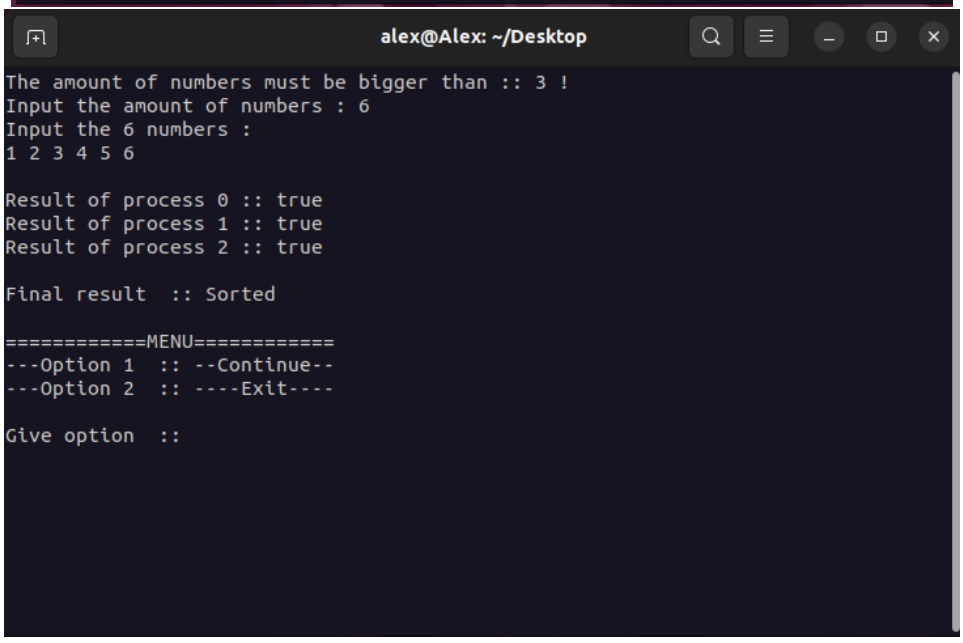
Εδώ όλες οι διεργασίες εκτός της πρώτης στέλνουν στην πρώτη διεργασία τα αποτελέσματά τους , η διεργασία μηδέν τυπώνει τα δικά της αποτελέσματα και ελέγχει αν έχουν βρεί κάποιο λάθος . Ο λόγος που τυπώνεται πάντα το λάθος που βρέθηκε πρώτο είναι διότι οι τιμές έχουν αρχικοποιηθεί σε -1 ,έτσι αν γίνει κάποια αλλαγή η συνθήκη (pos==-1 && tmppos!=-1) δεν θα είναι πια αληθής αρα δεν θα αλλάξει και το τελικό αποτέλεσμα.

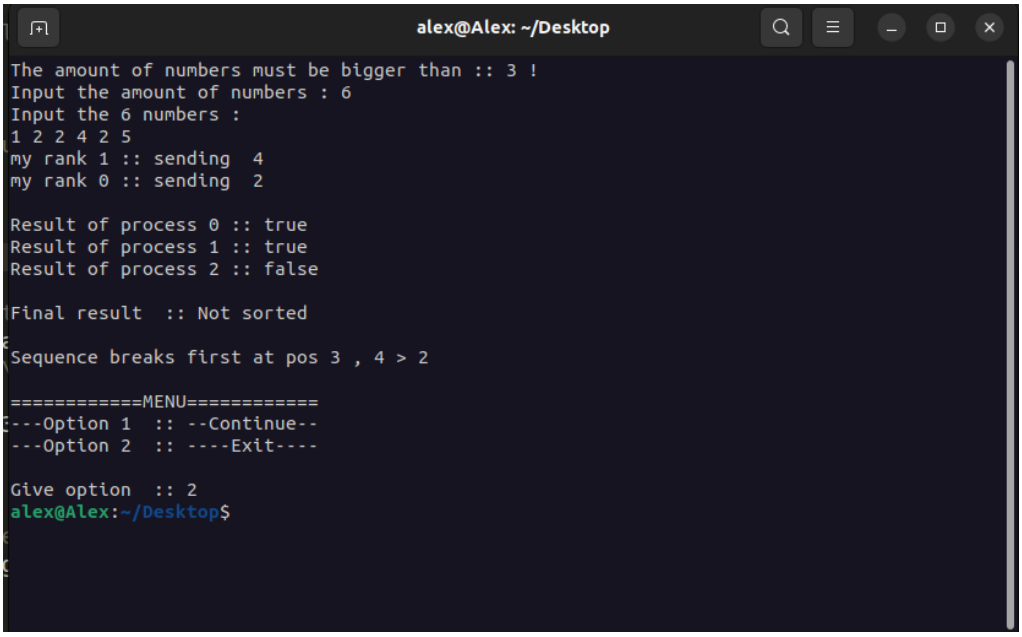
Τέλος , η διεργασία 0 εκτυπώνει το μενού και ρωτάει τον χρήστη αν θέλει να επαναλάβει την διαδικασία η να τερματίσει.

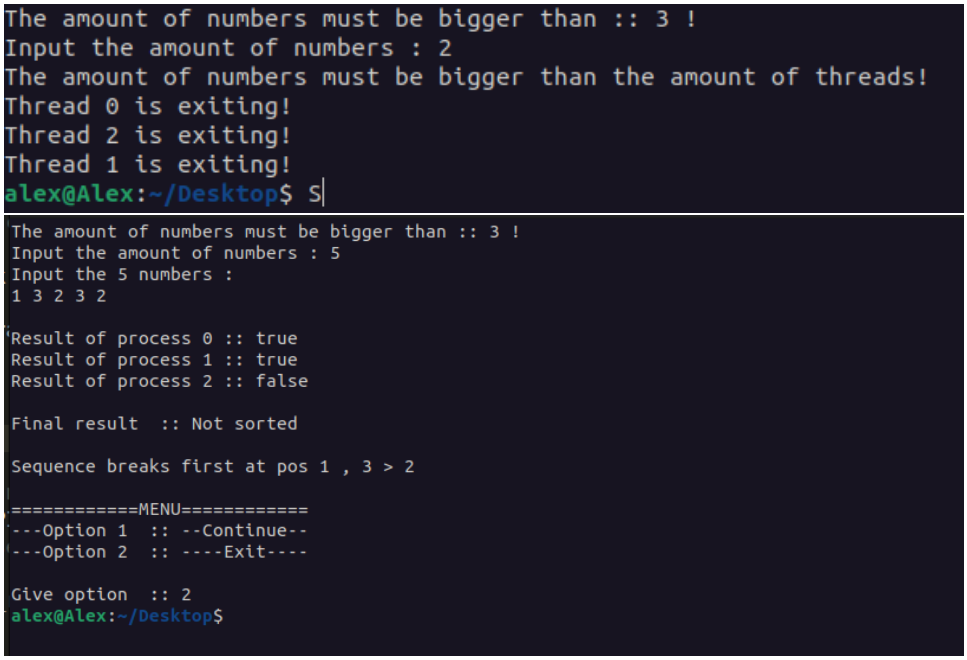
```
    if(my_rank==0){
        k=0;
        while(1){
            if(k>0) printf("There is no such option such as %d , please give again!\n",ch);
            printf("\n============MENU============\n");
            printf("---Option 1  :: --Continue--\n");
            printf("---Option 2  :: ----Exit----\n");
            printf("\nGive option  :: ");
            scanf("%d",&ch);
            if(!(ch==1 || ch==2)){
                k++;
               system("clear");                /* printing menu and getting the choice to c
            }else break;
        }
        for(target=1;target<p;target++)
            MPI_Send(&ch, 1, MPI_INT,target , tag3, MPI_COMM_WORLD);
        if(ch==2){
            MPI_Finalize();    /*exiting the programm if ch==2*/
        }
    }else{
        MPI_Recv(&ch, 1, MPI_INT,0, tag3, MPI_COMM_WORLD, &status);
        if(ch==2){
            MPI_Finalize();
        }
    }

}while(ch!=2);
```

ΕΝΔΕΙΚΤΙΚΑ ΤΡΕΞΙΜΑΤΑ






alex@Alex: ~/Desktop
The amount of numbers must be bigger than :: 3 !
Input the amount of numbers : 6
Input the 6 numbers :
1 2 2 4 2 5
my rank 1 :: sending 4
my rank 0 :: sending 2
Result of process 0 :: true
Result of process 1 :: true
Result of process 2 :: false
Final result :: Not sorted
Sequence breaks first at pos 3 , 4 > 2
============MENU============
---Option 1 :: --Continue--
---Option 2 :: ----Exit----
Give option :: 2
alex@Alex:~/Desktop$


The amount of numbers must be bigger than :: 3 !
Input the amount of numbers : 2
The amount of numbers must be bigger than the amount of threads!
Thread 0 is exiting!
Thread 2 is exiting!
Thread 1 is exiting!
alex@Alex:~/Desktop$ S
The amount of numbers must be bigger than :: 3 !
Input the amount of numbers : 5
Input the 5 numbers :
1 3 2 3 2
Result of process 0 :: true
Result of process 1 :: true
Result of process 2 :: false
Final result :: Not sorted
Sequence breaks first at pos 1 , 3 > 2
============MENU============
---Option 1 :: --Continue--
---Option 2 :: ----Exit----
Give option :: 2
alex@Alex:~/Desktop$