

ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΠΑΡΑΛΛΗΛΟ ΥΠΟΛΟΓΙΣΜΟ

Εργασία 1

ΠΑΝΤΕΛΕΙΜΩΝ ΠΡΩΙΟΣ
ice18390023
5ο Εξάμηνο
ice18390023@uniwa.gr

Τμήμα Ε2 Δευτέρα 13:00-14:00



ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
UNIVERSITY OF WEST ATTICA

Υπεύθυνοι καθηγητές

ΜΑΜΑΛΗΣ ΒΑΣΙΛΗΣ
ΙΟΡΔΑΝΑΚΗΣ ΜΙΧΑΛΗΣ

Τμήμα Μηχανικών και Πληροφορικής Υπολογιστών
2020-2021

Περιεχόμενα

1	Παράλληλος έλεγχος ταξινόμησης	1
1.1	Παραδείγματα από τρεξίματα ορθής λειτουργίας και σημειώσεις	1

1 Παράλληλος έλεγχος ταξινόμησης

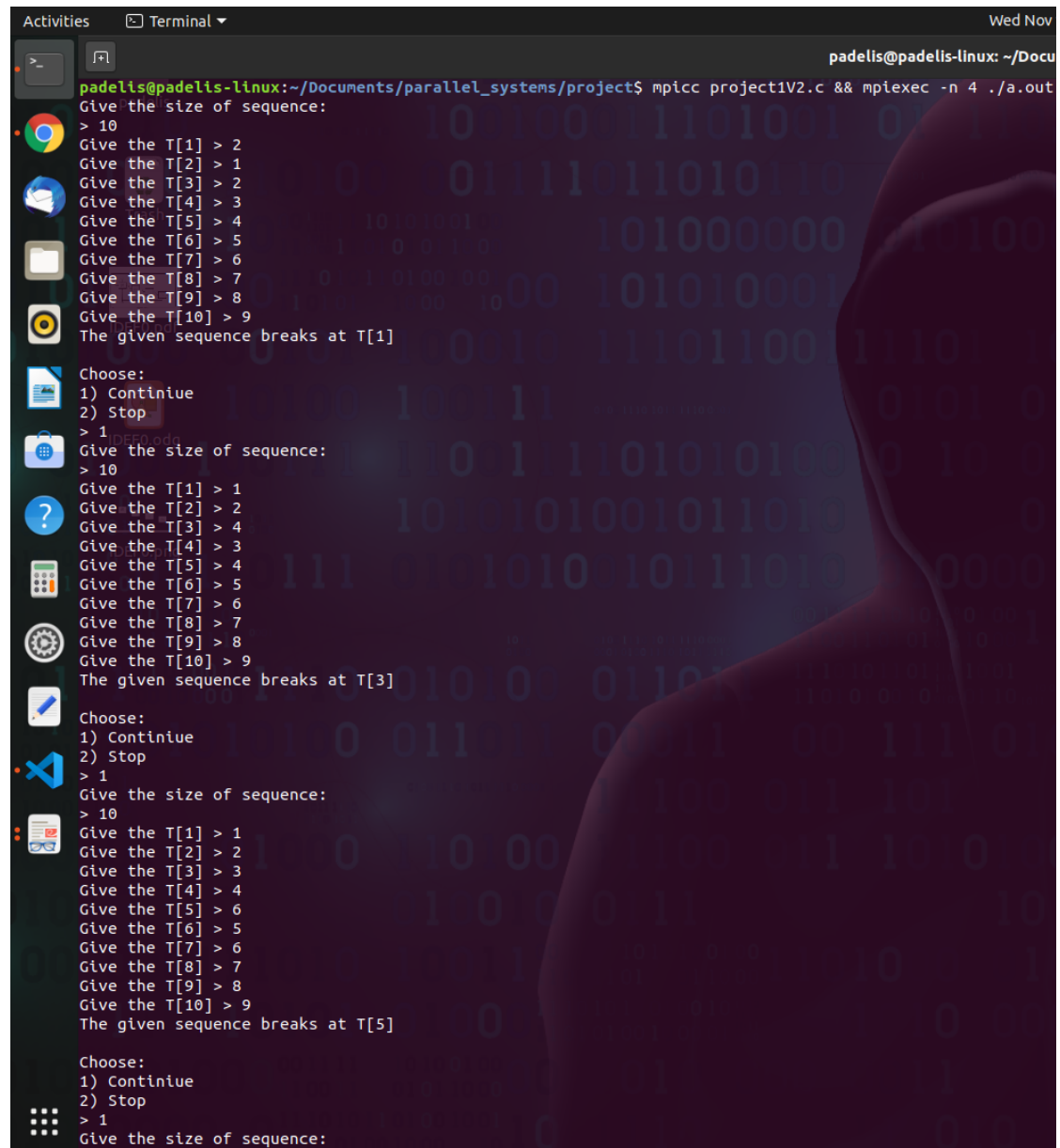
Σε περίπτωση που κάτι δεν βγάζει νόημα, στα σχόλια του κώδικα, ευθύνεται το ~~X₃~~TeX. Επίσης τα σχόλια στα ελληνικά δεν αναγνωρίζονται, για αυτό είναι μαύρα και όχι πράσινα. Είναι προτιμότερο, τα σχόλια να αναγνωστούν, από το αρχείο C.

1.1 Παραδείγματα από τρέξιματα ορθής λειτουργίας και σημειώσεις

Ο κώδικας εκτελείται κανονικά χωρίς πρόβλημα. Θα μπορούσαν να γίνουν μερικές αλλαγές, όπως σε κάποια σημεία, που χρησιμοποιούν `MPI_Send()`, να αντικατασταθούν με `MPI_Isend()`, αν και δεν έχει διαφορά για μικρά μεγέθη, επειδή η MPI το φροντίζει αποθηκεύοντας το σε έναν buffer, μέχρις ότου να αναγνωσθεί.

Ένα πρόβλημα είναι, πως τα δεδομένα πρέπει να είναι τουλάχιστον όσοι και οι επεξεργαστές. Αυτό θα μπορούσε να επιλυθεί με 2 if συνθήκες, εντός της `if(rank==0)` και η άλλη στο `else`, έτσι ώστε να ενημερώνει αν θα παραλάβουν δεδομένα. Δεν ήταν ζητούμενο οπότε παραλήφθηκε και δεν υλοποιήθηκε.

Οι εικόνες που ακολουθούν, είναι με την σειρά, ένα ενδεικτικό τρέξιμο, της ορθής λειτουργίας του προγράμματος. Επίσης, υπάρχουν δύο ίδιοι κώδικες, ο κώδικας 1.1 είναι με σχόλια όπως, στο αρχείο C και ο κώδικας 1.2, είναι χωρίς σχόλια.



```
Activities Terminal Wed Nov
padelis@padelis-linux: ~/Documents/parallel_systems/project$ mpicc project1V2.c && mplexec -n 4 ./a.out
Give the size of sequence:
> 10
Give the T[1] > 2
Give the T[2] > 1
Give the T[3] > 2
Give the T[4] > 3
Give the T[5] > 4
Give the T[6] > 5
Give the T[7] > 6
Give the T[8] > 7
Give the T[9] > 8
Give the T[10] > 9
The given sequence breaks at T[1]
Choose:
1) Continue
2) Stop
> 1
Give the size of sequence:
> 10
Give the T[1] > 1
Give the T[2] > 2
Give the T[3] > 4
Give the T[4] > 3
Give the T[5] > 4
Give the T[6] > 5
Give the T[7] > 6
Give the T[8] > 7
Give the T[9] > 8
Give the T[10] > 9
The given sequence breaks at T[3]
Choose:
1) Continue
2) Stop
> 1
Give the size of sequence:
> 10
Give the T[1] > 1
Give the T[2] > 2
Give the T[3] > 3
Give the T[4] > 4
Give the T[5] > 6
Give the T[6] > 5
Give the T[7] > 6
Give the T[8] > 7
Give the T[9] > 8
Give the T[10] > 9
The given sequence breaks at T[5]
Choose:
1) Continue
2) Stop
> 1
Give the size of sequence:
```

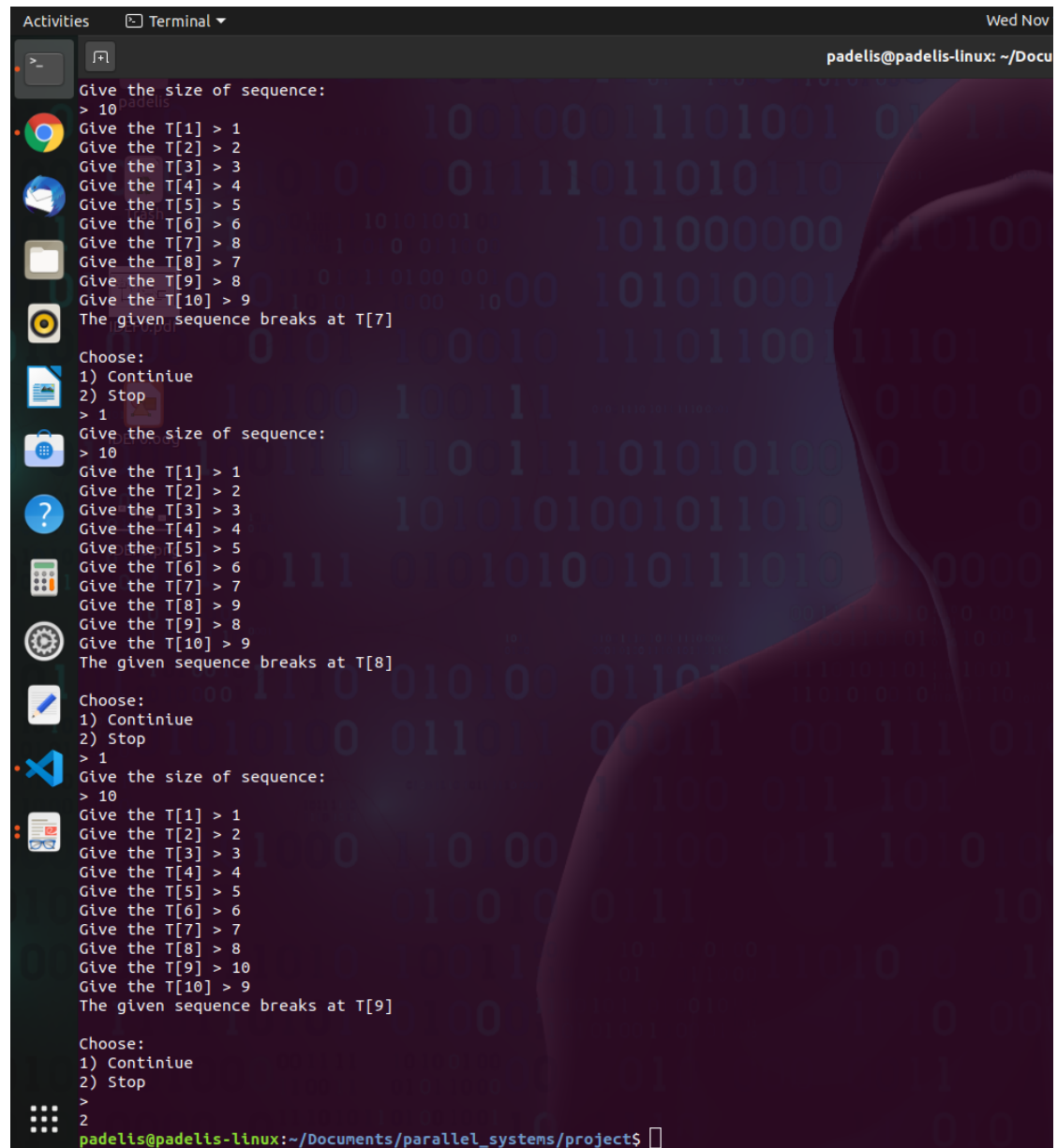
```
Activities Terminal Wed Nov
padelis@padelis-linux: ~/Docu

Choose: elis
1) Continue
2) Stop
> 1
Give the size of sequence:
> 10
Give the T[1] > 1
Give the T[2] > 2
Give the T[3] > 3
Give the T[4] > 4
Give the T[5] > 5
Give the T[6] > 7
Give the T[7] > 6
Give the T[8] > 7
Give the T[9] > 8
Give the T[10] > 9
The given sequence breaks at T[6]

Choose:
1) Continue
2) Stop
> 1
Give the size of sequence:
> 10
Give the T[1] > 1
Give the T[2] > 2
Give the T[3] > 3
Give the T[4] > 4
Give the T[5] > 5
Give the T[6] > 6
Give the T[7] > 8
Give the T[8] > 7
Give the T[9] > 8
Give the T[10] > 9
The given sequence breaks at T[7]

Choose:
1) Continue
2) Stop
> 1
Give the size of sequence:
> 10
Give the T[1] > 1
Give the T[2] > 2
Give the T[3] > 3
Give the T[4] > 4
Give the T[5] > 5
Give the T[6] > 6
Give the T[7] > 7
Give the T[8] > 9
Give the T[9] > 8
Give the T[10] > 9
The given sequence breaks at T[8]

Choose:
```



```
Activities Terminal Wed Nov
padelis@padelis-linux: ~/Docu

Give the size of sequence:
> 10
Give the T[1] > 1
Give the T[2] > 2
Give the T[3] > 3
Give the T[4] > 4
Give the T[5] > 5
Give the T[6] > 6
Give the T[7] > 8
Give the T[8] > 7
Give the T[9] > 8
Give the T[10] > 9
The given Sequence breaks at T[7]

Choose:
1) Continue
2) Stop
> 1
Give the size of sequence:
> 10
Give the T[1] > 1
Give the T[2] > 2
Give the T[3] > 3
Give the T[4] > 4
Give the T[5] > 5
Give the T[6] > 6
Give the T[7] > 7
Give the T[8] > 9
Give the T[9] > 8
Give the T[10] > 9
The given sequence breaks at T[8]

Choose:
1) Continue
2) Stop
> 1
Give the size of sequence:
> 10
Give the T[1] > 1
Give the T[2] > 2
Give the T[3] > 3
Give the T[4] > 4
Give the T[5] > 5
Give the T[6] > 6
Give the T[7] > 7
Give the T[8] > 8
Give the T[9] > 10
Give the T[10] > 9
The given sequence breaks at T[9]

Choose:
1) Continue
2) Stop
> 2
padelis@padelis-linux: ~/Documents/parallel_systems/projects$
```

```
1 #include <stdio .h>
2 #include <stdlib .h>
3 #include "mpi.h"
4
5
6 int menu();
7 int read_size(int );
8 int *read_sequence(int );
9 int *create_malloc(int );
10 int check_local(int *, int );
11
12 int main(int argc, char** argv){
13
14     // δεν χρειάζονται όλες οι μεταβλητές, αλλά είναι για καλύτερη κατανόηση.
15     int rank;
16     int source, dest;
17     int *tag;
18     int N, p;
19     int *T;
20     int *addr;
21     int extra;
22     int size;
23     int tmp;
24     int i;
25     int broke;
26     int rank_broke;
27     int nextT;
28
29     // ===== MPI Initalized =====
30
31     // Η MPI_Init, επιστρέφει MPI_SUCCESS, στην επιτυχία.
32     // Στην μεταβλητή tmp, εκχωρείται η επιστρεφόμενη τιμή της MPI_Init και
33     // η συνθήκη, ελέγχει αν η tmp δεν είναι MPI_SUCCESS, έτσι ώστε να πράξει
34     // καταλλήλως, αλλιώς να συνεχίσει κανονικά.
35
36     tmp = MPI_Init( &argc, &argv);
37
38     if (tmp != MPI_SUCCESS){
39         perror("MPI initialization ");
40         exit(EXIT_FAILURE);
41     }
42
43     MPI_Status status;
44
45     // κάθε processor μαθαίνει τον αριθμό του, για τον Communicator MPI_COMM_WORLD
46     // και εκχωρείται στην rank.
47     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
48     MPI_Comm_size(MPI_COMM_WORLD, &p); // Πόσοι υπάρχουν, υπο τον MPI_COMM_WORLD
49                                         // θα είναι ο ίδιος αριθμός για όλους του MPI_COMM_WORLD
50
51     tag = create_malloc(p); // γίνεται allocation, για p θέσεις υπάρχει( μια παραπάνω του εαυτού του)
52
53     // Επανάληψη, για πολλαπλές φορές ελέγχων.
54     while(1){
55
56         // Αρχικοποιούμε όλα τα tags με 0.
57         for (i = 0; i < p; i++){
58             tag[i]=0;
59         }
60
61         // Η συνθήκη θα είναι αληθής μόνο για τον processor που έχει rank == 0.
62         if ( rank == 0 ){
```

```

63
64 // Καλεί την συνάρτηση και επιστρέφει τον αριθμό των ακαεραίων που θα διαβάσει.
65 N = read_size(p);
66
67 // Επιστρέφει την διεύθυνση των ακεραίων που αναγνώστηκαν.
68 T = read_sequence(N);
69
70 size = N/p; // Το μικρότερο μέγεθος που θα έχουν όλοι.
71 addr = T + size; // Δείχνει απο ποία θέση και μετά θα πάρουν οι άλλοι δεδομένα.
72 extra = N%p; // Πόσοι θα πάρουν +1 παραπάνω.
73 if ( extra ){ // Αν υπάρχουν έξτρα, τότε σίγουρα ένα θα κρατήσει η rank 0
74 // και θα αυξήσει κατά 1 την θέση, όπου θα ξεκινήσει ο
75 addr += 1; // διαμιρασμός τον υπολοίπων.
76 }
77 // πχ.. αν N = 8 και p = 3, η αρχικοποίηση της addr είναι T+2+1
78 // πχ.. αν N = 8 και p = 4, η αρχικοποίηση της addr είναι T+2
79
80 // Αυτή η επανάληψη, είναι για όσους θα λάβουν +1 παραπάνω.
81 // Το dest αρχικοποιείται με 1, διότι βρισκόμαστε ήδη εντός του rank 0
82 // και πάντα ο rank 0 θα παίρνει 1 επιπλέον, αν υπάρχει κάποιος.
83 // Οπότε αρχίζει και ενημερώνει απο το 1 και μετά.
84 // Στην περίπτωση που το extra είναι 1, τότε η συνθήκη dest < extra
85 // θα είναι ψευδής, διότι 1 < 1 δεν ισχύει, οπότε δεν θα γίνει καμία
86 // επανάληψη, παρά μόνο η αρχικοποίηση dest = 1.
87 for ( dest = 1; dest < extra; dest++){
88
89 // Στην πρώτη γραμμή κώδικα απο κάτω, αυξάνουμε κατά ένα το tag του processor
90 // που θα στείλουμε τα δεδομένα. Στην επόμενη γραμμή, εκχωρούμε στην tmp, το
91 // πλήθος των δεδομένων που θα στείλουμε σε( αργότερο μάθημα, μάθαμε για τον
92 // MPI_Probe).
93 // Τέλος, στην τρίτη γραμμή, καλούμε την MPI_Send η( οποία θα μπορούσε να
94 // είναι MPI_Isend, δηλαδή non-blocking, αν και για λίγα δεδομένα δεν μπλοκάρει
95 // ούτε η MPI_Send, γιατί αντιγράφονται σε έναν local buffer μέχρι να
96 // διαβαστούν). Το πρώτο όρισμα, είναι η διεύθυνση που θα διαβάσει τα δεδομένα,
97 // το δεύτερο είναι το μέγεθος των δεδομένων, το τρίτο όρισμα είναι ο τύπος των
98 // δεδομένων, το τέταρτο όρισμα είναι ο αριθμός του processor που θα στείλουμε
99 // τα δεδομένα, το πέμπτο όρισμα είναι ο αριθμός του tag και το τελευταίο
100 // είναι ο communicator. είναι( σπατάλη, αλλά μάθαμε αργότερα για την MPI_Probe)
101
102 tag[dest]++;
103 tmp = size+1;
104 MPI_Send( &tmp, 1, MPI_INT, dest, tag[dest], MPI_COMM_WORLD);
105
106
107 // Αυξάνουμε κατα ένα, το tag του dest. Ξανά καλούμε την MPI_Send όπου(
108 // πάλι θα μπορούσε να είναι MPI_Isend), αλλά με πρώτο όρισμα την διεύθυνση
109 // του στοιχείου στον πίνακα που θα αρχίσει να παίρνει τα δεδομένα μεγέθους
110 // size+1 το( +1 είναι λόγω του έξτρα παραπάνω). Και έπειτα, αυξάνει την
111 // μεταβλητή addr όπου( είναι η διεύθυνση που θα ξεκινήσει να διαβάζει το επόμενο),
112 // κατά το πλήθος των δεδομένων που διάβασε, δηλαδή size+1.
113 // Η addr, έχει αρχικοποιηθεί απο πριν.
114 // πχ.. αν N = 8 και p = 3, η αρχικοποίηση της addr είναι T+2+1,
115 // οπότε θα γίνει αύξηση με τα δεδομένα που διαβάστηκαν, για να δίνουμε στην
116 // διεύθυνση του επόμενου στοιχείου προς ανάγνωση.
117 tag[dest]++;
118 MPI_Send(addr, size+1, MPI_INT, dest, tag[dest], MPI_COMM_WORLD);
119 addr += (size+1);
120 }
121
122
123 // Όπως και με στην προηγούμενη επανάληψη, έτσι και εδώ ισχύουν τα ίδια με τρείς
124 // διαφορές. Δεν γίνεται αρχικοποίηση της dest, διότι συνεχίζει να στέλνει δεδομένα

```



```

125 // στους υπόλοιπους, δηλαδή από τον αριθμό και μετά, όπου η συνθήκη του προηγούμενου
126 // βρόγχου ήταν ψευδή η( μεταβλητή dest αρχικοποιείται έτσι και αλλιώς με 1, ακόμα
127 // και αν δεν γίνει ούτε μια επανάληψη του προηγούμενου βρόγχου). Η πρώτη MPI_Send
128 // στέλνει size αντί για tmp, όπου ήταν size+1 η tmp. Και πλέων, την addr την
129 // αυξάνουμε κατά size, αντί για size+1.
130 for ( ; dest < p; dest++){
131
132     tag[dest]++;
133     MPI_Send(&size, 1, MPI_INT, dest, tag[dest], MPI_COMM_WORLD);
134
135     tag[dest]++;
136     MPI_Send(addr, size, MPI_INT, dest, tag[dest], MPI_COMM_WORLD);
137     addr += size;
138 }
139 // Σε αυτό το σημείο, η addr δίνει εκτός πίνακα κατά ένα στοιχείο και δεν την
140 // χρειαζόμαστε για κάτι άλλο.
141 // ===== msg has been sent =====
142
143
144 // Εκχωρούμε στην addr, την διεύθυνση της T.
145 addr = T;
146
147 // Αν το υπόλοιπο δεν είναι 0, τότε σίγουρα ένα παραπάνω το έχει η rank 0.
148 // Οπότε αυξάνουμε την μεταβλητή size κατά 1 η( μεταβλητή size ήταν αρχικοποιημένη
149 // με N/p).
150 if (N%p) size++;
151
152 // Κάνουμε allocate το μέγεθος των στοιχείων για την rank 0 θα( μπορούσαμε να
153 // κάνουμε realloc ) και εκχωρούμε την διεύθυνση στην T.
154 T = create_malloc( size );
155
156
157 // Αντιγράφουμε τα στοιχεία μεγέθους size, από την addr στην T.
158 // Πλέον η T έχει τα στοιχεία τα οποία αντιστοιχούν στην rank 0.
159 for (i = 0; i < size; i++){
160     T[i] = addr[i];
161 }
162
163 // Απελευθέρωση της δεσμευμένης μνήμης, όλων των στοιχείων που δώθηκαν.
164 free(addr);
165
166 // Στην μεταβλητή broke, εκχωρείται η επιστρεφόμενη τιμή της check_local.
167 // Η check_local δέχεται ως πρώτο όρισμα, διεύθυνση πίνακα ακεραίων και
168 // ως δεύτερο, το μέγεθος του πίνακα. Κάνει έλεγχο ταξινόμησης και επιστρέφει,
169 // -1 για επιτυχία και σε κάθε άλλη περίπτωση, τον αριθμό του στοιχείου.
170 broke = check_local( T, size );
171
172 // Αν η broke δεν είναι -1, τότε στην rank_broke εκχωρείται η rank, δηλαδή 0.
173 // Η μεταβλητή rank_broke, περιέχει τον αριθμό του rank που βρήκε αταξινόμητα
174 // στοιχεία.
175 if ( broke != -1) rank_broke = rank;
176
177
178 // Η συνθήκη ελέγχει, αν δεν είναι μόνο ένας ο processor.
179 if ( p != 1){
180
181     // Στην μεταβλητή source, εκχωρείται η τιμή του επόμενου processor πάντα( για
182     // τον communicator MPI_COMM_WORLD), όπου θα του στείλει απάντηση.
183     source = rank + 1; // 0 + 1 = 1
184
185     // Αυξάνουμε κατά ένα το tag του processor source με δείκτη source
186     tag[source]++;

```

```

187 // Σε αυτην την MPI_Recv, γίνεται λήψη απο τον επόμενο processor, του
188 // στοιχείου προς σύγκριση.
189 // Στην MPI_Recv, περνάμε ως πρώτο όρισμα την διεύθυνση μιας ακέραιας μεταβλητής
190 // για να αντιγραφούν τα δεδομένα σε αυτή.
191 // το δεύτερο είναι το μέγεθος των δεδομένων, το τρίτο όρισμα είναι ο τύπος των
192 // δεδομένων, το τέταρτο όρισμα είναι ο αριθμός του processor όπου θα λάβουμε
193 // τα δεδομένα, το πέμπτο όρισμα είναι ο αριθμός του tag, το έκτο είναι ο
194 // communicator και τελευταίο είναι MPI_Status struct για πληροφορίες.
195 MPI_Recv(&nextT, 1, MPI_INT, source, tag[source], MPI_COMM_WORLD, &status);
196
197
198 // Γίνεται έλεγχος, αν το τελευταίο στοιχείου του πίνακα είναι μεγαλύτερο
199 // απο τον αριθμό που λάβαμε στην μεταβλητή nextT του δηλαδή( του πρώτου
200 // στοιχείου του επόμενου processor) και αν η broke είναι -1 δηλαδή( δεν
201 // έχει βρεθεί ήδη στοιχείο όπου να σπάει η ταξινόμηση). Αν η συνθήκη είναι
202 // αληθής, τότε στην broke εκχωρείται ο δείκτης του στοιχείου που βρέθηκε
203 // η αταξινόμηση και στην rank_broke εκχωρείται ο αριθμός του rank στο οποίο
204 // βρέθηκε η αταξινόμηση.
205 if ( T[size-1] > nextT && broke == -1){
206     broke = size-1;
207     rank_broke = rank;
208 }
209
210 // Η rank 0, λαμβάνει μηνύματα απο όλους τους επεξεργαστές, έτσι ώστε
211 // να ενημερωθεί για αταξινόμηση.
212 for ( source = 1; source < p; source++){
213
214     // Αυξάνεται κατα 1 η τιμή του tag του source.
215     tag[source]++;
216
217     // Γίνεται λήψη ενός ακεραίου, ο οποίος θα είναι, είτε -1 αν δεν υπήρξε
218     // αταξινόμηση, είτε το στοιχείο στο οποίο βρέθηκε η αταξινόμηση.
219     MPI_Recv(&tmp, 1, MPI_INT, source, tag[source], MPI_COMM_WORLD, &status);
220
221     // Γίνεται έλεγχος, αν η tmp είναι διαφορη του -1 και η broke ίση με -1.
222     // Αν η tmp είναι -1, τότε ήταν τα ξινομημένα και αν η broke δεν είναι
223     // -1, τότε έχει βρεθεί στοιχείο ήδη για το οποίο σπάει η ταξινόμηση.
224     // Αν η συνθήκη είναι αληθής, τότε στην broke εκχωρείται ο αριθμός της
225     // tmp και στην rank_broke ο αριθμός της source
226     if ( tmp != -1 && broke == -1){
227         broke = tmp;
228         rank_broke = source;
229     }
230 }
231 } /* if (p != 1) */
232
233 // Αν η broke είναι -1, τότε όλα ήταν ταξινομημένα.
234 if ( broke == -1){
235     printf("The given sequence is sorted\n");
236 }
237 else {
238
239     // Στην tmp, εκχωρούμε το γινόμενο της ακέραιας διαίρεσης N/p και
240     // της rank_broke. Δηλαδή, θεωρικά τον πρώτο δείκτη, όπου θα ξεκινούσε
241     // τον έλεγχο ο rank_broke, αν δεν υπήρχαν έξτρα δεδομένα
242     // Στην συνθήκη, ελέγχουμε αν η rank_broke είναι μικρότερη της extra.
243     // Αν είναι αληθής, τότε συναθροίζουμε την broke όπου( είναι ο τοπικός
244     // δείκτης του στοιχείου, που έσπασε η ταξινόμηση), με το άθροισμα της tmp
245     // όπου( είναι τα σίγουρα στοιχεία που έχουν ελεγχεί πριν την rank_broke)
246     // και της rank_broke όπου( είναι τα παραπάνω στοιχεία). Αν ήταν ψευδής,
247     // τότε απλώς θα προσθέταμε τα extra που υπάρχουν.
248     // πχ.. έστω N=10 και p=4

```

```

249 // Για rank_broke = 0 και broke = 1, τότε tmp = 2*0 = 0 και broke += 0 + 0 = 1
250 // Για rank_broke = 1 και broke = 1, τότε tmp = 2*1 = 2 και broke += 2 + 1 = 4
251 // Για rank_broke = 2 και broke = 1, τότε tmp = 2*2 = 4 και broke += 4 + 2 = 7
252 // Για rank_broke = 3 και broke = 1, τότε tmp = 2*3 = 6 και broke += 6 + 3 = 9
253 tmp = N/p * rank_broke;
254 rank_broke < extra ? ( broke += (tmp + rank_broke)) : (broke += (tmp + N%p));
255 printf("The given sequence breaks at T[%d]\n", broke+1);
256 }
257
258 }
259 else {
260 // Όσοι δεν είναι rank 0.
261
262 // Το source είναι 0, γιατί όλοι έστω( και ένας) θα περιμένουν μήνυμα απο τον rank 0
263 source = 0;
264 tag[source]++; // Αυξάνεται το tag του source κατά 1.
265
266 // Παραλαμβάνουν όλοι, το μέγεθος των στοιχείων που θα λάβουν, στην μεταβλητή size
267 // μάθαμε( αργότερα για το MPI_Probe).
268 MPI_Recv( &size, 1, MPI_INT, source, tag[source], MPI_COMM_WORLD, &status);
269
270 // Γίνεται allocate για τα δεδομένα που θα ληφθούν.
271 T = create_malloc( size );
272
273 tag[source]++; // Αυξάνεται το tag του source κατά 1.
274
275 // Το πρώτο όρισμα, είναι η διεύθυνση που έγινε allocate, για να αντιγραφούν
276 // τα δεδομένα. Το δεύτερο όρισμα, είναι το μέγεθος που μπορούν να ληφθούν.
277 // Το τρίτο όρισμα, είναι ο τύπος των δεδομένων που θα ληφθούν. Το τέταρτο
278 // όρισμα είναι απο που θα ληφθούν, το οποίο θα είναι πάντα 0. Το πέμπτο
279 // όρισμα, είναι το tag. Το έκτο όρισμα, είναι ο communicator. Και το τελευταίο
280 // είναι μια δομή MPI_Status για παραπάνω πληροφορίες.
281 MPI_Recv( T, size, MPI_INT, source, tag[source], MPI_COMM_WORLD, &status);
282
283 // ===== msg has been received =====
284
285
286 // Στην μεταβλητή broke, εκχωρείται η επιστρεφόμενη τιμή της check_local.
287 // Η check_local δέχεται ως πρώτο όρισμα, διεύθυνση πίνακα ακεραίων και
288 // ως δεύτερο, το μέγεθος του πίνακα. Κάνει έλεγχο ταξινόμησης και επιστρέφει,
289 // -1 για επιτυχία και σε κάθε άλλη περίπτωση, τον αριθμό του στοιχείου.
290 broke = check_local( T, size );
291
292 // Στην μεταβλητή dest, αρχικοποιείται η τιμή του προηγούμενου processor
293 dest = rank - 1;
294 tag[dest]++; // Αυξάνεται το tag του source κατά 1.
295
296 // Η MPI_Send όπου( θα μπορούσε να είναι MPI_Isend), στέλνει τον πρώτο της
297 // ακέραιο, για να ελεγχεί απο τον dest. Το πρώτο όρισμα είναι η διεύθυνση T,
298 // όπου είναι το πρώτο στοιχείο του πίνακα και το δεύτερο όρισμα, είναι το μέγεθος
299 // των στοιχείων, που θα σταλούν.
300 MPI_Send( T, 1, MPI_INT, dest, tag[dest], MPI_COMM_WORLD);
301
302 // Γίνεται έλεγχος, αν το rank είναι διάφορο του p-1, δηλαδή δεν είναι ο τελευταίος
303 if ( rank != p-1){
304
305 // Το source, είναι ο επόμενος processor απο το παρον
306 source = rank + 1;
307 tag[source]++; // Αυξάνεται το tag του source κατά 1.
308
309 // Στην μεταβλητή nextT, αντιγράφετε ο αριθμός που στάλθηκε.
310 MPI_Recv( &nextT, 1, MPI_INT, source, tag[source], MPI_COMM_WORLD, &status);

```

```

311
312 // Γίνεται έλεγχος, αν το τελευταίο στοιχείο του processor, είναι μεγαλύτερο
313 // από αυτό που στάλθηκε και αν η broke είναι ίση με -1, επειδή μπορεί ήδη
314 // να υπήρξε αταξινόμηση στον πίνακα του processor. Αν η συνθήκη είναι αληθής,
315 // τότε στην broke εκχωρείται ο δείκτης τους τελευταίου στοιχείου.
316 if ( T[size-1] > nextT && broke == -1) broke = size -1;
317 }
318
319 // Το dest είναι 0, γιατί το μήνυμα θα σταλεί στον rank 0
320 dest = 0;
321 tag[dest]++; // Αυξάνεται το tag του source κατά 1.
322
323 // Ενημερώνει τον rank 0, αν υπήρξε σωστή ταξινόμηση ή όχι και σε ποίο στοιχείο
324 MPI_Send(&broke, 1, MPI_INT, dest, tag[dest], MPI_COMM_WORLD);
325
326 } /* else except rank 0 */
327
328 // Απελευθέρωση της δεσμευμένης μνήμης, όλων των στοιχείων που δώθηκαν.
329 free(T);
330
331 // Το περιεχόμενο των παρακάτω συνθηκών, θα μπορούσαν να ήταν στο τέλος της
332 // προηγούμενη συνθήκη, μαζί με μια free(T) πρώτα, εντός της κάθε συνθήκης.
333
334 // Η συνθήκη ελέγχει αν το rank είναι 0.
335 if (rank == 0){
336
337 // Στην tmp, εκχωρείται η επιστρεφόμενη τιμή, της συνάρτησης menu().
338 // Η πιστρεφόμενη τιμή, είναι ο αριθμός που δώθηκε από το stdin, 1 για συνέχεια
339 // και 2 για έξοδο. Σε περίπτωση διαφορετικής τιμής του 1, πάλι θα συνεχίσει.
340 tmp = menu();
341
342 // Γίνεται ενημέρωση σε όλους αν( υπάρχουν), με την τιμή που δώθηκε.
343 for ( dest = 1; dest < p; dest++){
344 tag[dest]++;
345 MPI_Send(&tmp, 1, MPI_INT, dest, tag[dest], MPI_COMM_WORLD);
346 }
347
348 // Αν η τιμή είναι 2, τότε βγαίνει από τον βρόγχο, αλλιώς συνεχίζει
349 if ( tmp == 2) break;
350 // Ίσος να ήταν ποίο σωστό, να βγαίνει σε οποιαδήποτε τιμή και να συνεχίζει μόνο
351 // στο 1, δηλαδή:
352 // tmp == 1 ? continue:break;
353 }
354 else {
355
356 // Το source είναι 0, γιατί από εκεί θα γίνει η ενημέρωση.
357 source = 0;
358 tag[source]++;
359
360 // Στην tmp αντιγράφεται ο ακέραιος που θα στείλει η rank 0.
361 MPI_Recv(&tmp, 1, MPI_INT, source, tag[source], MPI_COMM_WORLD, &status);
362
363 // Αν η τιμή που αντιγράφηκε στην tmp είναι 2, τότε βγαίνει από τον βρόγχο,
364 // αλλιώς συνεχίζει.
365 if ( tmp == 2) break;
366 // Ίσος να ήταν ποίο σωστό, να βγαίνει σε οποιαδήποτε τιμή και να συνεχίζει μόνο
367 // στο 1, δηλαδή:
368 // tmp == 1 ? continue:break;
369 }
370
371 } /* while */
372

```

```
373 // Απελευθέρωση της δεσμευμένης μνήμης, όλων των στοιχείων που δώθηκαν.
374 free(tag);
375
376 // Τέλος της παραλληλίας και της χρήσης των συναρτήσεων MPI.
377 MPI_Finalize();
378
379 return 0;
380 }
381
382 // Η συνάρτηση read_size, έχει ως παράμετρο το μέγεθος των processor. Εκτυπώνει στο stdout
383 // κατάλληλο μήνυμα και περιμένει για έναν ακέραιο αριθμό. Ο αριθμός πρέπει να είναι
384 // μεγαλύτερος ή ίσος με τους επεξεργαστές, αλλιώς θα εκτυπωθεί κατάλληλο μήνυμα και θα
385 // ξανά ζητηθεί αριθμός, μέχρις ότου να είναι μεγαλύτερος ή ίσος από τον αριθμό των
386 // επεξεργαστών. Τέλος, θα επιστραφεί αυτός ο αριθμός.
387 int read_size(int p){
388     int N;
389
390     do{
391         printf("Give the size of sequence:\n> ");
392         scanf("%d", &N);
393         if ( N < p){
394             printf("Size MUST be greater or equal to procecces\n");
395         }
396     }while(N < p);
397
398     return N;
399 }
400
401
402 // Η συνάρτηση read_sequence, δέχεται ως παράμετρο, το μέγεθος των στοιχείων προς ανάγνωση
403 // από το stdin. Επιστρέφει την διεύθυνση.
404 int *read_sequence(int N){
405
406     int i, tmp;
407
408     // Εκχωρεί στην ptr την επιστρεφόμενη διεύθυνση από την create_malloc
409     int *ptr = create_malloc(N);
410
411     // Κάνει N επαναλήψεις για την ανάγνωση όλων των δεδομένων από το stdin.
412     for ( i = 0; i < N; i++){
413         printf("Give the T[%d] > ", i+1);
414         scanf("%d", &tmp);
415         ptr[i] = tmp;
416     }
417     return ptr;
418 }
419
420
421 // Η συνάρτηση create_malloc, δέχεται ως παράμετρο το πλήθος των ακεραίων που θα γίνουν
422 // allocate, κάνει έλεγχο και επιστρέφει την διεύθυνση.
423 int *create_malloc(int len){
424
425     int *addr = NULL;
426
427     addr = (int *) malloc( len * sizeof(int) );
428
429     if (addr == NULL){
430         perror("Malloc error");
431         exit(EXIT_FAILURE);
432     }
433
434     return addr;
```

```
435 }
436 }
437
438 // Η συνάρτηση check_local, δέχεται ως πρώτη παράμετρο, την διεύθυνση ενός ακεραίου πίνακα
439 // και ως δεύτερη παράμετρο, το μέγεθος του πίνακα.
440 int check_local(int *arr, int size){
441
442     int i;
443
444     // Γίνεται έλεγχος του πίνακα, αν είναι ταξινομημένος. Αν είναι, αληθής η συνθήκη εντός
445     // του βρόγχου, τότε επιστρέφει τον αριθμό i, δηλαδή το δείκτη του πίνακα.
446     for (i = 0; i < size-1 ; i++){
447         if ( arr[i] > arr[i+1] ) return i;
448     }
449
450     return -1;
451 }
452
453 // Η συνάρτηση menu, εμφανίζει κατάλληλο μήνυμα στο stdout και περιμένει έναν ακέραιο από το
454 // stdin, το οποίο επιστρέφει. Θα μπορούσε να γίνεται έλεγχος ορθότητας του αριθμού, μέσα
455 // σε έναν βρόγχο, δηλαδή do {...} while(option > 0 && option < 3);
456 int menu(){
457     int option;
458     printf("\nChoose:\n1) Continue\n2) Stop\n> ");
459     scanf("%d", &option);
460     return option;
461 }
```

Κώδικας 1.1: Έλεγχος ταξινόμησης

```
1 #include <stdio .h>
2 #include <stdlib .h>
3 #include "mpi.h"
4
5
6 int menu();
7 int read_size( int );
8 int *read_sequence( int );
9 int *create_malloc( int );
10 int check_local( int *, int );
11
12 int main(int argc, char** argv){
13
14     int rank;
15     int source, dest;
16     int *tag;
17     int N, p;
18     int *T;
19     int *addr;
20     int extra;
21     int size;
22     int tmp;
23     int i;
24     int broke;
25     int rank_broke;
26     int nextT;
27
28     // ===== MPI Initalized =====
29
30     tmp = MPI_Init( &argc, &argv);
31
32     if (tmp != MPI_SUCCESS){
33         perror("MPI initialization ");
34         exit(EXIT_FAILURE);
35     }
36
37     MPI_Status status;
38
39     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
40     MPI_Comm_size(MPI_COMM_WORLD, &p);
41
42     tag = create_malloc(p);
43
44     while(1){
45
46         for (i = 0; i < p; i++){
47             tag[i]=0;
48         }
49
50         if ( rank == 0 ){
51
52             N = read_size(p);
53
54             T = read_sequence(N);
55
56             size = N/p;
57             addr = T + size;
58             extra = N%p;
59             if ( extra ){
60
61                 addr += 1;
62             }
63         }
64     }
```

```

63
64     for ( dest = 1; dest < extra; dest++){
65
66         tag[dest]++;
67         tmp = size + 1;
68         MPI_Send( &tmp, 1, MPI_INT, dest, tag[dest], MPI_COMM_WORLD);
69
70         tag[dest]++;
71         MPI_Send(addr, size + 1, MPI_INT, dest, tag[dest], MPI_COMM_WORLD);
72         addr += (size + 1);
73     }
74
75
76     for ( ; dest < p; dest++){
77
78         tag[dest]++;
79         MPI_Send( &size, 1, MPI_INT, dest, tag[dest], MPI_COMM_WORLD);
80
81         tag[dest]++;
82         MPI_Send(addr, size, MPI_INT, dest, tag[dest], MPI_COMM_WORLD);
83         addr += size;
84     }
85     // ===== msg has been sent =====
86
87     addr = T;
88
89     if (N%p) size++;
90
91     T = create_malloc( size );
92
93
94     for (i = 0; i < size; i++){
95         T[i] = addr[i];
96     }
97
98     free(addr);
99
100     broke = check_local( T, size );
101
102     if ( broke != -1) rank_broke = rank;
103
104
105     if ( p != 1){
106
107         source = rank + 1; // 0 + 1 = 1
108
109         tag[source]++;
110
111         MPI_Recv( &nextT, 1, MPI_INT, source, tag[source], MPI_COMM_WORLD, &status);
112
113         if ( T[size-1] > nextT && broke == -1){
114             broke = size - 1;
115             rank_broke = rank;
116         }
117
118         for ( source = 1; source < p; source++){
119
120             tag[source]++;
121
122             MPI_Recv( &tmp, 1, MPI_INT, source, tag[source], MPI_COMM_WORLD, &status);
123
124             if ( tmp != -1 && broke == -1){

```



```

125         broke = tmp;
126         rank_broke = source;
127     }
128 }
129 } /* if (p != 1) */
130
131 if ( broke == -1){
132     printf("The given sequence is sorted\n");
133 }
134 else{
135
136     // πχ.. έστω N=10 και p=4
137     // Για rank_broke = 0 και broke = 1, τότε tmp = 2*0 = 0 και broke += 0 + 0 = 1
138     // Για rank_broke = 1 και broke = 1, τότε tmp = 2*1 = 2 και broke += 2 + 1 = 4
139     // Για rank_broke = 2 και broke = 1, τότε tmp = 2*2 = 4 και broke += 4 + 2 = 7
140     // Για rank_broke = 3 και broke = 1, τότε tmp = 2*3 = 6 και broke += 6 + 3 = 9
141     tmp = N/p * rank_broke;
142     rank_broke < extra ? ( broke += (tmp + rank_broke)) : (broke += (tmp + N%p));
143     printf("The given sequence breaks at T[%d]\n", broke+1);
144 }
145
146 }
147 else {
148     // Όσοι δεν είναι rank 0.
149
150     source = 0;
151     tag[source]++;
152
153     MPI_Recv( &size, 1, MPI_INT, source, tag[source], MPI_COMM_WORLD, &status);
154
155     T = create_malloc( size );
156
157     tag[source]++;
158
159     MPI_Recv( T, size, MPI_INT, source, tag[source], MPI_COMM_WORLD, &status);
160
161     // ===== msg has been received =====
162
163
164     broke = check_local( T, size );
165
166     dest = rank - 1;
167     tag[dest]++;
168
169     MPI_Send( T, 1, MPI_INT, dest, tag[dest], MPI_COMM_WORLD);
170
171     if ( rank != p-1){
172
173         source = rank + 1;
174         tag[source]++;
175
176         MPI_Recv( &nextT, 1, MPI_INT, source, tag[source], MPI_COMM_WORLD, &status);
177
178         if ( T[size-1] > nextT && broke == -1) broke = size -1;
179     }
180
181     dest = 0;
182     tag[dest]++;
183
184     MPI_Send( &broke, 1, MPI_INT, dest, tag[dest], MPI_COMM_WORLD);
185
186 } /* else except rank 0 */

```

```

187     free(T);
188
189
190     if (rank == 0){
191
192         tmp = menu();
193
194         for ( dest = 1; dest < p; dest++){
195             tag[dest]++;
196             MPI_Send(&tmp, 1, MPI_INT, dest, tag[dest], MPI_COMM_WORLD);
197         }
198
199         if ( tmp == 2) break;
200     }
201     else{
202
203         source = 0;
204         tag[source]++;
205
206         MPI_Recv(&tmp, 1, MPI_INT, source, tag[source], MPI_COMM_WORLD, &status);
207
208         if ( tmp == 2) break;
209     }
210
211     } /* while */
212
213     free(tag);
214
215     MPI_Finalize();
216
217     return 0;
218 }
219
220
221 int read_size(int p){
222     int N;
223
224     do{
225         printf("Give the size of sequence:\n> ");
226         scanf("%d", &N);
227         if ( N < p){
228             printf("Size MUST be greater or equal to procecces\n");
229         }
230     }while(N < p);
231
232     return N;
233 }
234
235
236 int *read_sequence(int N){
237
238     int i, tmp;
239
240     int *ptr = create_malloc(N);
241
242     for ( i = 0; i < N; i++){
243         printf("Give the T[%d] > ", i+1);
244         scanf("%d", &tmp);
245         ptr[i] = tmp;
246     }
247     return ptr;
248 }

```

```
249
250
251 int *create_malloc(int len){
252
253     int *addr = NULL;
254
255     addr = (int *) malloc( len * sizeof(int) );
256
257     if (addr == NULL){
258         perror("Malloc error");
259         exit(EXIT_FAILURE);
260     }
261
262     return addr;
263
264 }
265
266
267 int check_local(int *arr, int size){
268
269     int i;
270
271     for (i = 0; i < size-1 ; i++){
272         if ( arr[i] > arr[i+1] ) return i;
273     }
274
275     return -1;
276 }
277
278 int menu(){
279     int option;
280     printf("\nChoose:\n1) Continue\n2) Stop\n> ");
281     scanf("%d", &option);
282     return option;
283 }
```

Κώδικας 1.2: Έλεγχος ταξινόμησης (χωρίς σχόλια)