

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ ΥΠΟΛΟΓΙΣΤΩΝ

ΑΝΑΦΟΡΑ 1<sup>ης</sup> ΕΡΓΑΣΤΗΡΙΑΚΗΣ ΑΣΚΗΣΗΣ

---



Στοιχεία Ομάδας

- Αναγνωριστικό: oslab005
- Μέλος 1<sup>ο</sup>: Πέππας Μιχαήλ – Αθανάσιος, Α.Μ: 03121026
- Μέλος 2<sup>ο</sup>: Σαουνάτσος Ανδρέας, Α.Μ: 03121197
- Ημερομηνία Παράδοσης Αναφοράς: 22.04.2024

## ■ Ενότητα 1.1 – Ανάγνωση και εγγραφή αρχείων στη C και με τη βοήθεια κλήσεων συστήματος

Στη συγκεκριμένη άσκηση, ζητούμενο αποτελεί η υλοποίηση ενός προγράμματος που αναζητά πόσες φορές εμφανίζεται ένας χαρακτήρας, μέσα σε ένα αρχείο εισόδου, και γράφει το αποτέλεσμα σε ένα αρχείο εξόδου. Τα αρχεία και ο χαρακτήρας προς αναζήτηση δίνονται από το χρήστη, στη γραμμή εντολών, ενώ χρησιμοποιούμε αποκλειστικά κλήσεις συστήματος προκειμένου να επιτελέσουμε όλες τις ζητούμενες λειτουργίες. Επισημαίνουμε ότι τόσο στην άσκηση αυτή όσο και στις υπόλοιπες έχουμε συντάξει κώδικα, ο οποίος διαχειρίζεται όλα τα πιθανά λάθη και αντιδρά κατάλληλα, ενώ έχουμε υλοποιήσει (σε όλα τα αρχεία που εισάγουμε είσοδο από τη γραμμή εντολών) έλεγχο των εισαγόμενων ορισμάτων – παραμέτρων από τον χρήστη. Στον συμπιεσμένο φάκελο, ο κώδικας αυτός βρίσκεται στο αρχείο με όνομα «a1.1-system\_calls.c».

Ακολουθεί η τμηματική παρουσίαση του κώδικά μας, λογική που θα ακολουθήσουμε και σε όλες τις επόμενες ενότητες.

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/types.h>
4  #include <sys/stat.h>
5  #include <fcntl.h>
6  #include <string.h>
7
8  /* Theoroume tis eisodous argv[1],argv[2] ta onomata twn arxeiwn
9  pros diavasma kai grapsimo antistoixa. To argv[3][0] einai o
10 xarakthras pros anazhthsh */
11
12 int main(int argc, char**argv)
13 {
14
15     // arxikopoioume tis metavlhtes gia ta read kai write
16
17     int fd1, fd2;
18     int oflags, mode;
19     oflags=O_CREAT | O_WRONLY | O_TRUNC;
20     mode = S_IRUSR | S_IWUSR;
21     int count=0;
22     ssize_t rcnt;
23     char buff[1024];
24     ssize_t wcnt;
```

Στο συγκεκριμένο τμήμα του κώδικά μας, περιλαμβάνουμε τις κατάλληλες βιβλιοθήκες για τη συμπερίληψη των απαραίτητων συναρτήσεων και ορίζουμε – αρχικοποιούμε τις απαραίτητες μεταβλητές του προγράμματός μας. Αυτές είναι δύο περιγραφητές αρχείων (ένας για αυτό της ανάγνωσης και ένας για αυτό της εγγραφής), ο buffer για την ανάγνωση – εγγραφή, τα flags της εγγραφής (δημιουργία νέου αρχείου αν αυτό δεν υπάρχει, εγγραφή μόνο και καθαρισμός του πριν την εγγραφή, αντίστοιχα), το mode (που δίνει δικαίωμα εγγραφής και ανάγνωσης στον

χρήστη) και κάποιες μεταβλητές τύπου `ssize_t` για το αποτέλεσμα που επιστρέφουν οι κλήσεις συστήματος `read (rcnt)` και `write (wcnt)`.

```
26 /* elegxoume an ta orismata einai akriwvs 3 kai typwnoume to mnhma me write kai fd=0*/
27
28 if (argc!=4)
29 {
30     write(0,"The number of arguments must be exactly 3.\n",strlen("The number of arguments must be exactly 3.\n"));
31     return 1;
32 }
33
34 /* elegxoume an to trito orisma einai mono enas xarakthras
35 kai typwnoume to mnhma me write kai fd=0 */
36
37 if (argv[3][1]!='\0')
38 {
39     write(0,"The third argument must be only 1 character.\n",strlen("The third argument must be only 1 character.\n"));
40     return 1;
41 }
```

Στο παραπάνω τμήμα, ελέγχουμε εάν ο αριθμός των εισαγόμενων από τον χρήστη ορισμάτων είναι ακριβώς 4 (ένα του εκτελέσιμου ./, ένα του αρχείου ανάγνωσης, ένα του αρχείου εγγραφής και ένα του χαρακτήρα προς ψάξιμο) και σε περίπτωση που δεν είναι, τότε τυπώνει το παραπάνω μήνυμα και τερματίζει το πρόγραμμα. Ακολουθώντας, ελέγχουμε εάν το τελευταίο όρισμα είναι μόνο ένας χαρακτήρας (κάθε συμβολοσειρά στη C/C++ τερματίζει με 0, άρα ελέγχουμε εάν το δεύτερο στοιχείο του 3<sup>ου</sup> ορίσματος είναι το 0) και αν δεν είναι, τότε τυπώνουμε το αντίστοιχο μήνυμα (λάθους) και τερματίζουμε το πρόγραμμα. Σημειώνουμε ότι το γράψιμο στην οθόνη (του χρήστη) επιτελείται μέσω της `write`, με file descriptor 0.

```
44 /* ektelesh tou open kai gia ta 2 arxeia. To
45 prwto einai read only enw to deftero write only */
46
47 fd1=open(argv[1],O_RDONLY);
48 if (fd1==-1)
49 {
50     perror("open");
51     return 1;
52 }
53
54 fd2=open(argv[2], oflags,mode);
55 if (fd2==-1)
56 {
57     perror("open");
58     return 1;
59 }
```

Στο παραπάνω τμήμα ανοίγουμε (με την κλήση συστήματος `open`, που στην επιτυχία επιστρέφει τον περιγραφητή του αρχείου που ανοίξαμε) το πρώτο αρχείο μόνο για ανάγνωση και το δεύτερο αρχείο με τις παραμέτρους που ορίσαμε στην αρχή (`oflags` και `mode`), ενώ πραγματοποιούμε έλεγχο εάν amfότερα τα αρχεία

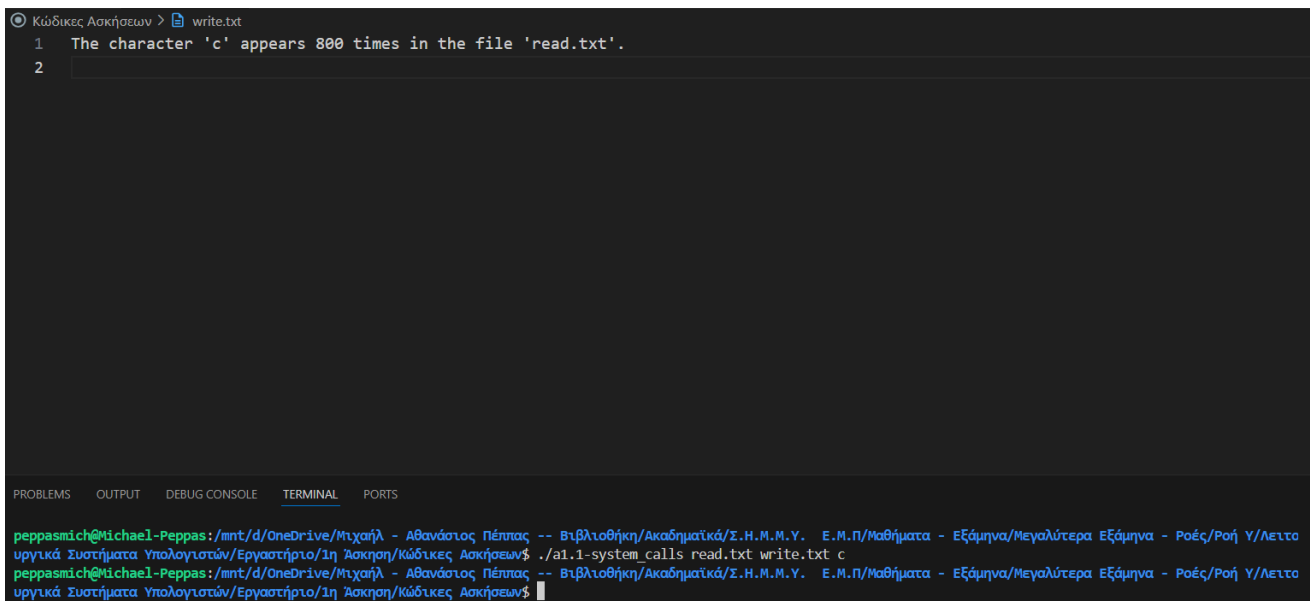
άνοιξαν ορθά ή εάν συνέβη κάποιο λάθος (-1), οπότε και τερματίζουμε το πρόγραμμα, εμφανίζοντας κατάλληλο μήνυμα λάθους στον χρήστη.

```
61  /*ekteloume thn diadikasia read, vazontas '\0' sto telos kathe diavasmatos
62  gia na mpouroume mpouroyme na elegxoume to telos tou buff */
63
64  for (;;)
65  {
66      rcnt = read(fd1, buff, sizeof(buff)-1);
67      if (rcnt==0) break;
68      if (rcnt==-1)
69      {
70          perror("read");
71          return 1;
72      }
73      buff[rcnt]= '\0';
74      int i=0;
75      while (buff[i]!='\0')
76      {
77          if (buff[i++]==argv[3][0]) count++;
78      }
79  }
```

Στο τμήμα αυτό, χρησιμοποιούμε την κλήση συστήματος read (που επιστρέφει πόσους χαρακτήρες διάβασε επιτυχώς) για να διαβάσουμε τους χαρακτήρες του αρχείου ανάγνωσης, ελέγχοντας για πιθανά λάθη (εάν επιστραφεί μηδέν φτάσαμε το EOF και φεύγουμε από τη λούπα, ενώ εάν επιστραφεί -1 συνέβη λάθος και έτσι τερματίζουμε το πρόγραμμα ενημερώνοντας κατάλληλα τον χρήστη). Κατόπιν, συγκρίνουμε έναν – έναν τους χαρακτήρες που αποθηκεύτηκαν στον buffer με τον ζητούμενο χαρακτήρα και κάθε φορά που τον συναντάμε αυξάνουμε τον μετρητή count κατά 1.

```
80
81  /* me thn sprintf metatrepoume thn frash pros egrafh se string (enw arxika eixe kai int kai char)
82  gia na mpoume na ektelesoume thn write */
83
84  int n = sprintf(buff, "The character '%c' appears %d times in the file '%s'.\n", argv[3][0], count, argv[1]);
85  wcnt = write(fd2, buff, n);
86  if (wcnt == -1)
87  {
88      perror("write");
89      return 1;
90  }
91  if (wcnt < n)
92  {
93      perror("write_space");
94      return 1;
95  }
96
97  close(fd1);
98  close(fd2);
99  return 0;
100 }
101
```

Τέλος, στο τμήμα αυτό, τυπώνουμε το αποτέλεσμα της αναζήτησης στο αρχείο εγγραφής, μέσω της κλήσης συστήματος write (η οποία επιστρέφει τα bytes που κατάφερε να γράψει) και ελέγχουμε εάν συνέβη κάποιο λάθος στην εγγραφή (-1), οπότε και τερματίζουμε το πρόγραμμα ενημερώνοντας κατάλληλα τον χρήστη, καθώς και εάν γράψαμε λιγότερο (π.χ. σε περίπτωση ανεπαρκούς χώρου) που και πάλι τερματίζουμε, εμφανίζοντας αντίστοιχο μήνυμα λάθους. Στο τέλος, κλείνουμε τα αρχεία που ανοίξαμε και τελειώνουμε το πρόγραμμα. Το αποτέλεσμα (στο αρχείο εγγραφής), φαίνεται ακολούθως:



The screenshot shows a code editor with a file named `write.txt`. The code in the editor is as follows:

```
1 The character 'c' appears 800 times in the file 'read.txt'.
2
```

Below the code editor, the terminal output is visible. It shows the command `./a1.1-system_calls read.txt write.txt c` being executed, and the output is the same text as in the code editor:

```
peppasmich@Michael-Peppas: /mnt/d/OneDrive/Μιχαήλ - Αθανάσιος Πέππας -- Βιβλιοθήκη/Ακαδημαϊκά/Σ.Η.Μ.Μ.Υ. Ε.Μ.Π/Μαθήματα - Εξάμηνα/Μεγαλύτερα Εξάμηνα - Ροές/Ροή Υ/Λειτουργικά Συστήματα Υπολογιστών/Εργαστήριο/1η Άσκηση/Κώδικες Ασκήσεων$ ./a1.1-system_calls read.txt write.txt c
peppasmich@Michael-Peppas: /mnt/d/OneDrive/Μιχαήλ - Αθανάσιος Πέππας -- Βιβλιοθήκη/Ακαδημαϊκά/Σ.Η.Μ.Μ.Υ. Ε.Μ.Π/Μαθήματα - Εξάμηνα/Μεγαλύτερα Εξάμηνα - Ροές/Ροή Υ/Λειτουργικά Συστήματα Υπολογιστών/Εργαστήριο/1η Άσκηση/Κώδικες Ασκήσεων$
```

## ■ Ενότητα 1.2 – Δημιουργία διεργασιών

Στη συγκεκριμένη ενότητα, απαντήσαμε στα πρώτα δύο ερωτήματα από κοινού, σε ένα αρχείο, το οποίο παρουσιάζουμε τμηματικά παρακάτω. Σκοπός μας είναι η δημιουργία μίας διεργασίας παιδί που θα χαιρετάει τον κόσμο, αναφέροντας το αναγνωριστικό της και το αναγνωριστικό του γονέα της, ενώ ο γονέας θα τυπώνει το αναγνωριστικό του παιδιού και θα περιμένει τον τερματισμό του. Ακόμα, ορίζουμε μια μεταβλητή, έστω  $x$ , στον γονέα πριν τη δημιουργία του παιδιού και αναθέτουμε διαφορετική τιμή στο γονέα και στο παιδί και τυπώνουμε την τιμή της. Στον συμπιεσμένο φάκελο, ο κώδικας αυτός βρίσκεται στο αρχείο με όνομα «a1.2-fork.c».

```
7  int main()
8  {
9
10 int x = 100;
11 pid_t p=fork();
12
13 if (p<0)
14 {
15     perror("fork");
16     exit(1);
17 }
18 else if (p==0) // Diergasia - paidi
19 {
20
21 //-----
22     printf("Hello world. My id (child) is %d and my parent's id is %d.\n",getpid(),getppid());
23     printf("%d\n", x);
24     x=50;
25     printf("%d\n", x);
26 //-----
27
28
29     exit(0);
30 }
```

Στο παραπάνω τμήμα, μετά τη συμπερίληψη των κατάλληλων βιβλιοθηκών, ορίζουμε μια global μεταβλητή  $x$  και την αρχικοποιούμε στην τιμή 100 και δημιουργούμε μια διεργασία – παιδί με την εντολή `fork()`, ελέγχοντας αν αυτή δημιουργήθηκε σωστά. Εάν προέκυψε σφάλμα ( $p<0$ ), τότε το πρόγραμμα τερματίζει και επιστρέφουμε το αντίστοιχο μήνυμα λάθους. Ακολούθως, το παιδί ( $p=0$ ) χαιρετάει τον κόσμο, αναφέροντας τον αναγνωριστικό του (εντολή `getpid()`) και το αναγνωριστικό του πατέρα του (`getppid()`). Επίσης, τυπώνει τη μεταβλητή  $x$  (προφανώς θα τυπωθεί 100, αφού ένα αντίγραφο του γονέα έχει δημιουργηθεί στο παιδί και το αντίγραφο αυτό περιλαμβάνει την τιμή  $x=100$ ), κατόπιν την αλλάζει σε 50 και την ξανατυπώνει (προφανώς θα τυπωθεί πλέον 50), ενώ στο τέλος τερματίζει επιτυχώς (`exit(0)`).

```

31 else // Diergasia - goneas
32 {
33 //-----
34     printf("%d\n", x);
35     x=80;
36     printf("%d\n", x);
37     printf("My child's id is %d.\n",wait(NULL));
38 //-----
39 }
40
41 return 0;
42 }
43

```

Στο τμήμα του κώδικα αυτό, εκτελείται ο πατέρας ( $p > 0$ ), ο οποίος τυπώνει τη  $x$  (προφανώς θα τυπωθεί 100, αφού περιλαμβάνει την αρχικοποιημένη μεταβλητή και το παιδί δεν τον επηρεάζει, αφού αυτό περιλαμβάνει αντίγραφο της κατάστασής του), κατόπιν τη θέτει σε 80 και την ξανατυπώνει (προφανώς θα τυπωθεί πλέον 80), ενώ στο τέλος επιστρέφει το αναγνωριστικό του παιδιού του, αφού το περιμένει, μέσω της `wait(NULL)`. Το πρόγραμμα τερματίζει επιτυχώς και τυπώνει τα κάτωθι:

```

peppasmich@Michael-Peppas:/mnt/d/OneDrive/Μιχαήλ - Αθανάσιος Πέππας -- Βιβλιοθήκη/Ακαδημαϊκά/Σ.Η.Μ.Υ. Ε.Μ.Π/Μαθήματα - Εξάμηνα/Μεγαλύτερα Εξάμηνα - Ροές/Ροή Υ/Λειτουργικά Συστήματα/Υπολογιστών/Εργαστήριο/1η Άσκηση/Κώδικες Ασκήσεων$ ./a1.2-fork
100
80
Hello world. My id (child) is 398 and my parent's id is 397.
100
50
My child's id is 398.

```

Προφανώς, ο γονέας και το παιδί εκτελούνται ταυτόχρονα, γι' αυτό οι αριθμοί δεν εμφανίζονται με τη σειρά που είναι στον κώδικα, ενώ σίγουρα το τελευταίο μήνυμα θα είναι πάντα του πατέρα με το αναγνωριστικό του παιδιού του, αφού το περιμένει να τερματίσει για να το τυπώσει.

Το 3<sup>ο</sup> ερώτημα της ενότητας αυτής είναι να αναθέσουμε στη διεργασία παιδί την αναζήτηση του χαρακτήρα στο αρχείο (όχι όμως τον υπόλοιπο χειρισμό των αρχείων). Έτσι, ο πατέρας ανοιγοκλείνει το αρχείο και το παιδί κάνει την επεξεργασία, ακριβώς όπως στην ενότητα 1.1. Στον συμπιεσμένο φάκελο, ο κώδικας αυτός βρίσκεται στο αρχείο με όνομα «a1.2-fork3.c».

```

10 int main(int argc, char** argv)
11 {
12
13     int fd1, fd2;
14     int oflags, mode;
15     oflags=O_CREAT | O_WRONLY | O_TRUNC;
16     mode=S_IRUSR | S_IWUSR;
17     int count=0;
18     ssize_t rcnt;
19     char buff[1024];
20     ssize_t wcnt;
21
22     /* elegxoume an ta orismata einai akrivws 3 kai typwnoyme to mynhma me write kai fd=0*/
23
24     if (argc!=4)
25     {
26         write(0,"The number of arguments must be exactly 3.\n",strlen("The number of arguments must be exactly 3.\n"));
27         return 1;
28     }
29
30     /* elegxoume an to trito orisma einai mono enas xarakthras
31     kai typwnoyme to mhnyma me write kai fd=0 */
32
33     if (argv[3][1]!='\0')
34     {
35         write(0,"The third argument must be only 1 character.\n",strlen("The third argument must be only 1 character.\n"));
36         return 1;
37     }

```

```

38 //----- parent opens files
39
40 fd1=open(argv[1],O_RDONLY);
41 if (fd1==-1)
42 {
43     perror("open");
44     return 1;
45 }
46
47 fd2=open(argv[2], oflags,mode);
48 if (fd2==-1)
49 {
50     perror("open");
51     return 1;
52 }
53

```

Στις δύο παραπάνω εικόνες (αφού συμπεριλάβαμε όλες τις απαραίτητες βιβλιοθήκες), ορίζουμε – αρχικοποιούμε τις μεταβλητές μας, κάνουμε τον έλεγχο των ορισμάτων από τον χρήστη και ο πατέρας ανοίγει τα αρχεία ανάγνωσης και εγγραφής, ελέγχοντας για πιθανά λάθη, όλα ακριβώς όπως στην ενότητα 1.1.



```

56     pid_t p = fork();
57
58     if (p<0) {perror("fork"); exit(1);}
59     else if (p == 0)
60     {
61         for (;;)
62         {
63             rcnt = read(fd1, buff, sizeof(buff)-1);
64             if (rcnt==0) break;
65             if (rcnt==-1) {perror("read"); return 1;}
66             buff[rcnt]= '\0';
67             int i=0;
68             while (buff[i]!='\0')
69             {
70                 if (buff[i++]==argv[3][0]) count++;
71             }
72         }
73
74         int n = sprintf(buff, "The character '%c' appears %d times in the file '%s'.\n", argv[3][0], count, argv[1]);
75         wcnt = write(fd2, buff, n);
76         if (wcnt == -1)
77         {
78             perror("write");
79             return 1;
80         }
81         if (wcnt < n)
82         {
83             perror("write_space");
84             return 1;
85         }

```

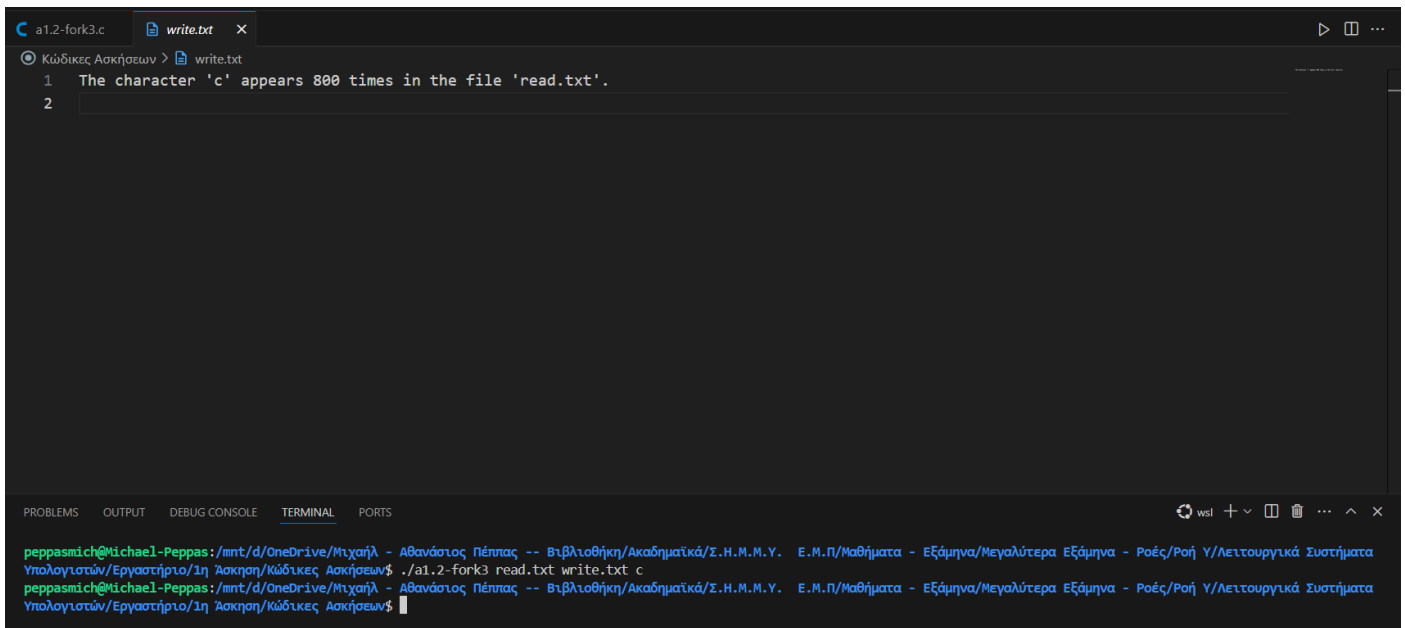
Στο τμήμα αυτό, κάνουμε το fork() και δημιουργούμε μια διεργασία – παιδί (ελέγχοντας εάν προέκυψε σφάλμα στη διαδικασία) και κατόπιν στο σώμα του παιδιού (p=0) αναθέτουμε την επεξεργασία του αρχείου (με κλήσεις συστήματος), ακριβώς όπως στον κώδικα της ενότητας 1.1 (για τον λόγο αυτό η επεξήγηση δεν θα επαναληφθεί), ενώ πάντοτε μεριμνούμε για όλα τα πιθανά λάθη που μπορεί να συμβούν. Στο παρακάτω τμήμα, εκτελείται το σώμα του πατέρα, ο οποίος κλείνει τα αρχεία:

```

86
87     exit(0);
88
89 }
90 else
91 {
92     wait(NULL);
93     close(fd1);
94     close(fd2);
95 }
96
97     return 0;
98 }
99

```

Η εκτέλεση του προγράμματος παρατίθεται ακολούθως:



The screenshot shows a Visual Studio Code editor with a file named `write.txt` open. The file contains two lines of C code:

```
1 The character 'c' appears 800 times in the file 'read.txt'.
2
```

Below the editor, the terminal window is active, showing the command `./a1.2-fork3 read.txt write.txt c` being executed. The terminal output shows the command being run and the program's execution path.

Προφανώς, το αποτέλεσμα είναι το ίδιο με αυτό της ενότητας 1.1.

Τέλος, στο 4<sup>ο</sup> ερώτημα δημιουργούμε μία διεργασία παιδί που θα εκτελεί τον κώδικα που μας δόθηκε στο Ερώτημα 1, μέσω της εντολής `execv`, η οποία παίρνει ως όρισμα το εκτελέσιμο της ενότητας 1.1 και τα ορίσματα και φορτώνει μια νέα εικόνα – διεργασία στο παιδί. Στον συμπιεσμένο φάκελο, ο κώδικας αυτός βρίσκεται στο αρχείο με όνομα «`a1.2-fork4.c`». Έτσι, συντάσσουμε το εξής πρόγραμμα:

```

1  #include <sys/wait.h>
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <sys/types.h>
5  #include <sys/stat.h>
6  #include <fcntl.h>
7  #include <stdlib.h>
8  #include <string.h>
9
10 int main (int argc, char** argv)
11 {
12
13     pid_t p = fork();
14
15     if (p<0) {perror("fork"); exit(1);}
16     else if (p==0)
17     {
18         if (execv("./a1.1-system_calls", argv) == -1)
19         {
20             perror("execv");
21             exit(1);
22         }
23         exit(0);
24     }
25     else
26         wait(NULL);
27
28     return 0;
29 }
30

```

Πάντοτε μεριμνούμε για τα πιθανά λάθη στη `fork()`, ενώ τώρα έχουμε βάλει και τον έλεγχο για την `execv` και σε περίπτωση λάθους, το πρόγραμμα τερματίζει, εμφανίζοντας το αντίστοιχο μήνυμα στον χρήστη. Το αποτέλεσμα της εκτέλεσης φαίνεται ακολούθως (προφανώς είναι ξανά το ίδιο με τις προηγούμενες 2 φορές), ενώ μετά την ορθή εκτέλεση, κάνουμε και μια επίδειξη του ελέγχου των εισαγόμενων ορισμάτων από τον χρήστη που υλοποιήσαμε στην ενότητα 1.1:

The image shows a Visual Studio Code editor window with a file named `write.txt` open. The file contains two lines of C code:

```
1 The character 'c' appears 800 times in the file 'read.txt'.
2
```

Below the editor, the **TERMINAL** panel is active, displaying the execution of the program `a1.2-fork4.c`. The terminal shows the user's commands and the program's output:

```
peppasmich@michael-Peppas:/mnt/d/OneDrive/Μιχαήλ - Αθανάσιος Πέππας -- Βιβλιοθήκη/Ακαδημαϊκά/Σ.Η.Μ.Μ.Υ. Ε.Μ.Π/Μαθήματα - Εξάμηνα/Μεγαλύτερα Εξάμηνα - Ροές/Ροή Υ/Λειτουργικά Συστήματα
Υπολογιστών/Εργαστήριο/1η Άσκηση/Κώδικες Ασκήσεων$ ./a1.2-fork4 read.txt write.txt c
peppasmich@michael-Peppas:/mnt/d/OneDrive/Μιχαήλ - Αθανάσιος Πέππας -- Βιβλιοθήκη/Ακαδημαϊκά/Σ.Η.Μ.Μ.Υ. Ε.Μ.Π/Μαθήματα - Εξάμηνα/Μεγαλύτερα Εξάμηνα - Ροές/Ροή Υ/Λειτουργικά Συστήματα
Υπολογιστών/Εργαστήριο/1η Άσκηση/Κώδικες Ασκήσεων$ ./a1.2-fork4 read.txt write.txt ca
The third argument must be only 1 character.
peppasmich@michael-Peppas:/mnt/d/OneDrive/Μιχαήλ - Αθανάσιος Πέππας -- Βιβλιοθήκη/Ακαδημαϊκά/Σ.Η.Μ.Μ.Υ. Ε.Μ.Π/Μαθήματα - Εξάμηνα/Μεγαλύτερα Εξάμηνα - Ροές/Ροή Υ/Λειτουργικά Συστήματα
Υπολογιστών/Εργαστήριο/1η Άσκηση/Κώδικες Ασκήσεων$ ./a1.2-fork4 read.txt write.txt c fifth
The number of arguments must be exactly 3.
peppasmich@michael-Peppas:/mnt/d/OneDrive/Μιχαήλ - Αθανάσιος Πέππας -- Βιβλιοθήκη/Ακαδημαϊκά/Σ.Η.Μ.Μ.Υ. Ε.Μ.Π/Μαθήματα - Εξάμηνα/Μεγαλύτερα Εξάμηνα - Ροές/Ροή Υ/Λειτουργικά Συστήματα
Υπολογιστών/Εργαστήριο/1η Άσκηση/Κώδικες Ασκήσεων$
```

## ■ Ενότητα 1.3 – Διεργασιακή επικοινωνία

Ζητούμενο της συγκεκριμένης ενότητας είναι η επέκταση του προγράμματος του ερωτήματος 2, ώστε να δημιουργεί P διεργασίες παιδιά (το P είναι ορισμένο σαν σταθερά στο πρόγραμμά μας, P = 5), οι οποίες θα αναζητούν παράλληλα τον χαρακτήρα στο αρχείο και η γονεϊκή διεργασία θα συλλέγει και θα τυπώνει το συνολικό αποτέλεσμα. Επίσης, όταν το πρόγραμμά μας δέχεται Control+C από το πληκτρολόγιο (δηλαδή το σήμα SIGINT) αντί να τερματίζει, τυπώνει τον συνολικό αριθμό διεργασιών που αναζητούν το αρχείο εκείνη τη χρονική στιγμή. Στον συμπιεσμένο φάκελο, ο κώδικας αυτός βρίσκεται στο αρχείο με όνομα «a1.3-comm.c».

```
10
11 // Prints the active children of the process
12 short unsigned int c_counter = 0, result = 0, P = 5;
13 int fd[2];
14 void child_counter (int signum)
15 {
16     // Printing the new signal
17     close(fd[1]);
18     for (short unsigned int i = 0; i < P; i++)
19     {
20         short unsigned int temp2 = 0;
21         if (read(fd[0], &temp2, sizeof(temp2)) == 0) break;
22         result += temp2;
23         c_counter--;
24     }
25
26     char buff_s[60];
27     int ns = sprintf(buff_s, "Control_C pressed: the number of active children is: %d.", c_counter);
28     ssize_t wcnt = write(0, buff_s, ns);
29     if (wcnt == -1) // Write error
30     {
31         perror("write");
32         exit(1);
33     }
34     if (wcnt < ns)
35     {
36         perror("write_space");
37         exit(1);
38     }
39 }
```

Στο τμήμα αυτό, μετά τη συμπερίληψη όλων των απαραίτητων βιβλιοθηκών, έχουμε ορίσει τις global μεταβλητές c\_counter (στην οποία αποθηκεύουμε τον αριθμό των παιδιών που αναζητούν τον χαρακτήρα στο αρχείο, τη στιγμή που θα δεχτούμε το σήμα SIGINT), result (στην οποία αποθηκεύουμε το συνολικό αποτέλεσμα της αναζήτησης του χαρακτήρα από όλες τις παράλληλες διεργασίες), τον αριθμό των διεργασιών (P) και το pipe (πίνακας int fd[2]). Ακολούθως, έχουμε ορίσει τη συνάρτηση που επιτελεί τη ζητούμενη διαδικασία όταν πατηθεί Control+C από τον χρήστη, η οποία κλείνει το άκρο εγγραφής του pipe (fd[1]) και διαβάζει από το άκρο ανάγνωσης (fd[0]) όσα στοιχεία υπάρχουν. Στην επόμενη εικόνα, θα γίνει

φανερó ότι κάθε φορά που γεννιέται ένα παιδί από το for loop, η μεταβλητή `c_counter` αυξάνεται κατά 1, επομένως εδώ μετράμε πόσα παιδιά έχουν τερματίσει την αναζήτηση στο αρχείο και έχουν επιστρέψει το αποτέλεσμα στο `pipe`, το οποίο και προσθέτουμε στη μεταβλητή `result` (μέσω του `temp2`) αφού γίνεται `pop` από το `pipe` και κατόπιν αφαιρούμε από τον συνολικό αριθμό των παιδιών που έχουν δημιουργηθεί αυτά που έχουν τελειώσει. Έτσι, η μεταβλητή `c_counter` περιέχει μόνο αυτά που τρέχουν τη δεδομένη χρονική στιγμή (δημιουργημένα – όσα τελείωσαν). Το αποτέλεσμα τυπώνεται στην οθόνη (0 ως file descriptor) και μεριμνούμε για πιθανά λάθη, όπως και τις προηγούμενες φορές.

```
41 int main(int argc, char** argv)
42 {
43     // Signal Handler
44     struct sigaction sa;
45     sigset_t sigset;
46     sa.sa_handler = child_counter;
47     sa.sa_flags = SA_RESTART;
48     memset(&sa.sa_restorer, 0, sizeof sa.sa_restorer);
49     memset(&sa.sa_mask, 0, sizeof sa.sa_mask);
50     if (sigaction(SIGINT, &sa, NULL) < 0)
51     {
52         perror("sigaction");
53         exit(1);
54     }
55
56     int f2;
57     int oflags = O_CREAT | O_WRONLY | O_TRUNC, mode = S_IRUSR | S_IWUSR;
```

Στο παραπάνω τμήμα, χρησιμοποιούμε την εντολή `sigaction`, ώστε να αλλάξουμε τη λειτουργία του σήματος `SIGINT` στη ζητούμενη, δηλαδή τη συνάρτηση `child_counter` μέσω του handler, μηδενίζοντας παράλληλα τις απαραίτητες θέσεις μνήμης με το `memset`, μεριμνώντας για την ορθή λειτουργία της `sigaction`, ώστε σε αντίθετη περίπτωση το πρόγραμμα να τερματίσει και να τυπώσει το αντίστοιχο μήνυμα λάθους. Ο λόγος που χρησιμοποιήσαμε τη `sigaction` και τη `memset` είναι για να λύσουμε ένα πρόβλημα που αντιμετωπίσαμε, κατά το οποίο μετά από μία μόνο χρήση του αλλαγμένου `SIGINT`, η λειτουργία του επαναφερόταν αυτόματα στο default. Τέλος, ορίζουμε έναν file descriptor και τις σημαίες/modes για την επεξεργασία των αρχείων μας, όπως και στις προηγούμενες ασκήσεις, τμήμα που ήδη εξηγήσαμε αναλυτικά.

```

59 // Check if number of arguments is proper
60 if (argc!=4)
61 {
62     write(0,"The number of arguments must be exactly 3.\n",
63         strlen("The number of arguments must be exactly 3.\n"));
64     return 1;
65 }
66
67 // Check if the last argument is a single character
68 if (argv[3][1]!='\0')
69 {
70     write(0,"The third argument must be only 1 character.\n",
71         strlen("The third argument must be only 1 character.\n"));
72     return 1;
73 }

```

Στο τμήμα αυτό, γίνεται ο γνωστός έλεγχος για τα εισαγόμενα από τον χρήστη ορίσματα, που έχει επεξηγηθεί ήδη πλήρως και έχει παρουσιαστεί στην ενότητα 1.2 με εικόνες.

```

74
75 // Pipe could not be established
76 if (pipe(fd) < 0)
77 {
78     perror("pipe");
79     exit(1);
80 }
81
82 // Error opening the file to write
83 f2 = open(argv[2], oflags, mode);
84 if (f2 == -1)
85 {
86     perror("open");
87     exit(1);
88 }

```

Στο παραπάνω τμήμα του κώδικά μας, δημιουργούμε το pipe, μέσω του οποίου επικοινωνούν ο γονέας και τα παιδιά, ελέγχοντας αν αυτό δημιουργήθηκε σωστά, ενώ συγχρόνως ανοίγουμε το αρχείο εγγραφής με τις παραμέτρους που ορίσαμε στην αρχή, ελέγχοντας για πιθανό λάθος κατά το άνοιγμα.

```

90     for (short unsigned int i = 0; i < P; i++)
91     {
92         c_counter++;
93         pid_t p = fork();
94         if (p < 0) // Error in the fork
95         {
96             perror("fork");
97             exit(1);
98         }
99         else if (p == 0) // Child proc
100        {
101            close(fd[0]);
102
103            // Signal Handler
104            struct sigaction sa;
105            sigset_t sigset;
106            sa.sa_handler = SIG_IGN;
107            memset(&sa.sa_restorer, 0, sizeof sa.sa_restorer);
108            memset(&sa.sa_flags, 0, sizeof sa.sa_flags);
109            memset(&sa.sa_mask, 0, sizeof sa.sa_mask);
110            if (sigaction(SIGINT, &sa, NULL) < 0)
111            {
112                perror("sigaction");
113                exit(1);
114            }

```

Στο παραπάνω for loop, το οποίο εκτελείται P φορές, δημιουργούνται P διεργασίες – παιδιά, τα οποία αναζητούν παράλληλα τον ζητούμενο χαρακτήρα στο αρχείο και κάθε φορά που δημιουργείται ένα νέο παιδί αυξάνουμε τη μεταβλητή c\_counter κατά 1 (τώρα βλέπε ξανά την παρουσίαση του πρώτου τμήματος, της συνάρτησης child\_counter) και ελέγχουμε εάν προέκυψε κάποιο σφάλμα. Εάν όλα πήγαν καλά, τότε στο σώμα του παιδιού κλείνουμε το άκρο ανάγνωσης (αφού αυτό αργότερα θα γράψει το αποτέλεσμα της αναζήτησης στο pipe) και ακολουθούμε την ίδια διαδικασία με την συνάρτηση child\_counter, ώστε να του περάσουμε τη νέα πληροφορία για το σήμα SIGINT.



```

116 int f1 = open(argv[1], O_RDONLY);
117 if (f1 == -1)
118 {
119     perror("open");
120     exit(1);
121 }
122 ssize_t rcnt;
123 char buff[1];
124 short unsigned int count = 0;
125 lseek(f1, i, SEEK_SET);
126 while (1)
127 {
128     rcnt = read(f1, buff, 1);
129     if (rcnt == 0) break;
130     if (rcnt == -1)
131     {
132         perror("read");
133         exit(1);
134     }
135     if (buff[0] == argv[3][0]) count++;
136     lseek(f1, P-1, SEEK_CUR);
137 }
138 write(fd[1], &count, sizeof(count));
139 close(f1);
140 exit(0);
141 }
142 else ;
143 }

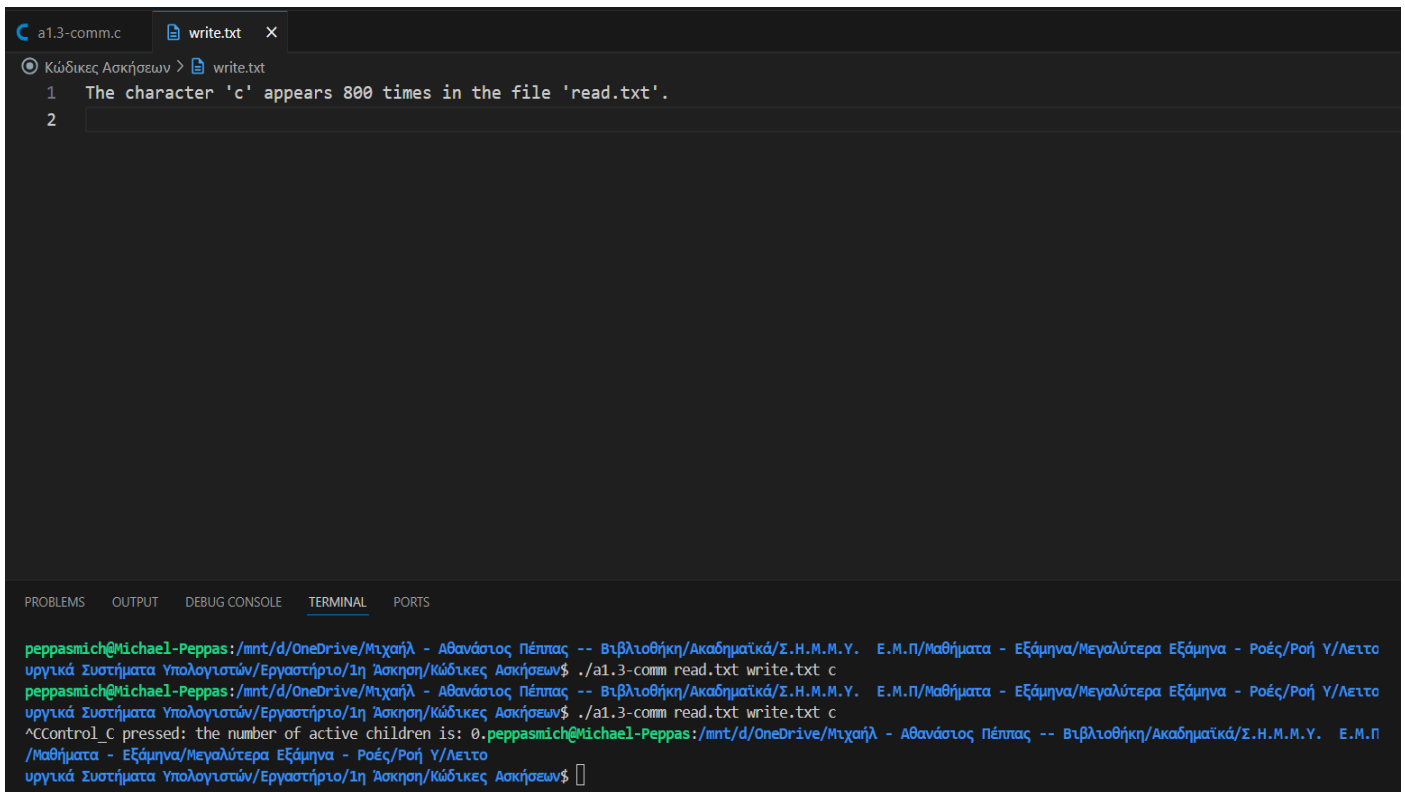
```

Έτσι, εδώ ανοίγουμε το αρχείο ανάγνωσης (ελέγχοντας για πιθανό λάθος κατά το άνοιγμα) και ορίζουμε έναν buffer ανάγνωσης (ενός χαρακτήρα μόνο) και μια μεταβλητή count, όπου αποθηκεύεται το αποτέλεσμα της αναζήτησης του κάθε παιδιού. Οι μεταβλητές αυτές ορίστηκαν μέσα στην εικόνα του παιδιού, ώστε να είναι από την αρχή ατομικές. Η λογική της αναζήτησης τώρα είναι η εξής: κάθε παιδί ξεκινάει από τη θέση i, μέσω της εντολής lseek (SEEK\_SET = πήγαινε στη θέση i) και κάνει ανάγνωση ενός χαρακτήρα τη φορά (ελέγχοντας για λάθη ανάγνωσης ή εάν φτάσαμε το EOF : rcnt = 0) και τον συγκρίνει με τον ζητούμενο χαρακτήρα (argv[3][0]) και αν είναι ο ζητούμενος, τότε αυξάνει τον μετρητή count κατά 1. Κατόπιν, μετατοπίζεται κατά P-1 θέσεις από το σημείο που βρίσκεται, μέσω της lseek (SEEK\_CUR = μετατόπιση κατά P-1 από το σημείο που βρίσκεται εκείνη τη στιγμή) και επαναλαμβάνει τη διαδικασία, μέχρι να φτάσει στο τέλος του αρχείου. Δηλαδή, κάθε παιδί ξεκινάει από μία συγκεκριμένη θέση και κάθε φορά διαβάζει έναν χαρακτήρα,

μετατοπισμένο κατά P από την τελευταία του ανάγνωση (η read ήδη κάνει μετατόπιση κατά 1 και έπειτα η lseek κατά P-1 ακόμα). Έτσι, τα παιδιά (P στο πλήθος) θα διαβάσουν με τον τρόπο αυτό όλους τους χαρακτήρες παράλληλα, με κατάλληλες διαδοχικές μετατοπίσεις. Στο τέλος, κάθε παιδί γράφει το αποτέλεσμα της αναζήτησης στο pipe (fd[1]) και κλείνει το αρχείο, τερματίζοντας επιτυχώς. Ο πατέρας δεν κάνει κάτι.

```
147     short unsigned int temp;
148     // Calculating the result
149     close(fd[1]);
150     for (short unsigned int i = 0; i < P; i++)
151     {
152         if (read(fd[0], &temp, sizeof(temp)) == 0) break;
153         result += temp;
154         c_counter--;
155     }
156
157     // Printing the result
158     char buff_r[60];
159     int n = sprintf(buff_r, "The character '%c' appears %d times in the file '%s'.\n", argv[3][0], result, argv[1]);
160     ssize_t wcnt = write(f2, buff_r, n);
161     if (wcnt == -1) // Write error
162     {
163         perror("write");
164         return 1;
165     }
166     if (wcnt < n)
167     {
168         perror("write_space");
169         return 1;
170     }
171
172     close(f2);
173
174     return 0;
175 }
```

Τέλος, ο πατέρας κλείνει το άκρο εγγραφής και διαβάζει από το άκρο ανάγνωσης ένα – ένα τα στοιχεία που έχουν τοποθετήσει τα παιδιά και τα περνάει μέσω της μεταβλητής temp στο συνολικό αποτέλεσμα result. Κάθε φορά μειώνει και τον αριθμό των παιδιών που είναι ενεργά (αφού προφανώς έχει συλλέξει το αποτέλεσμα της αναζήτησής του και άρα αυτό έχει τερματίσει επιτυχώς). Μετά, τυπώνουμε, ως γνωστόν, το αποτέλεσμα της συνολικής αναζήτησης στην οθόνη (ελέγχοντας για λάθη εγγραφής) και κλείνουμε και το αρχείο εγγραφής. Το πρόγραμμα τερματίζει επιτυχώς. Κάτωθι, φαίνεται το αποτέλεσμα της εκτέλεσής του (και του νέου σήματος SIGINT) και προφανώς είναι το ίδιο με όλα τα προηγούμενα:



The screenshot shows a VS Code editor with a file named `write.txt` open. The file contains the following C code:

```
1 The character 'c' appears 800 times in the file 'read.txt'.
2
```

Below the editor, the **TERMINAL** panel shows the execution of the program. The user runs the command `./a1.3-comm read.txt write.txt c`. The output shows the program's execution, including a message from the C library: `^CControl_C pressed: the number of active children is: 0.`

**Σημείωση:** Παρόλο που δεν φαίνεται στις φωτογραφίες με τον κώδικα, στα αρχεία κώδικα που συμπεριλαμβάνουμε στον συμπιεσμένο φάκελο, κάθε φορά που κάνουμε τον γνωστό έλεγχο των εισαγόμενων ορισμάτων από τον χρήστη έχουμε προσθέσει χειρισμό πιθανών λαθών από τη `write` και τυπώνουμε το αντίστοιχο μήνυμα λάθος (απλά δεν μπορούσαμε να αλλάξουμε όλες τις φωτογραφίες από την αρχή...).

**Σ.Η.Μ.Μ.Υ. Ε.Μ.Π.**  
**Απρίλιος 2024**