ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Άσκηση 1
Εισαγωγή στο περιβάλλον προγραμματισμού

Λειτουργικά Συστήματα

Εργαστηρική Ομάδα Β02

Βαβουλιώτης Γεώργιος
ΑΜ : 03112083
7ο εξάμηνο

Γιαννούλας Βασίλειος
ΑΜ : 03112117
7ο εξάμηνο

Ακ. Έτος 2015-2016

1.1 Σύνδεση με αρχείο αντικειμένων

Στην άσκηση αυτή καλούμαστε να δημιουργήσουμε ένα εκτελέσιμο αρχείομε όνομα zing, το οποίο θα καλεί τη συνάρτηση zing(). Η συνάρτηση zing() δηλώνεται στο αρχείο zing.h και για τον λόγο αυτό το πρώτο πράγμα που κάνουμε είναι να αντιγράψουμε στον κατάλογο εργασίας μας το αρχείο zing.h. Το ίδιο κάνουμε και με το αρχείο zing.o το οποίο είναι ένα object file και μας είναι αναγκαίο για να κάνουμε το linking(σύνδεση) του αρχείου zing.o με ένα άλλο αρχείο(θα εξηγηθεί παρακάτω). Και τα δυο αρχεία σύμφωνα με την εκφώνηση υπάρχουν στο κατάλογο /home/oslab/code/zing και η αντιγραφή αυτών στο κατάλογο εργασίας μας γίνεται με τις εξής εντολές:

```
cp  /home/oslab/code/zing/zing.o  ~
cp  /home/oslab/code/zing/zing.h  ~
```

Στη συνέχεια καλούμαστε να δημιουργήσουμε ένα αρχείο κώδικα main.c, το οποίο θα το χρησιμοποιήσουμε για να καλέσουμε τη συνάρτηση zing(). Ο κώδικας του main.c φαίνεται παρακάτω :

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

int main()
{
  zing();
  return 0;
}
```

Για να παράγουμε το αρχείο αντικειμένων main.o, μεταγλωτίζουμε το αρχείο main.c με χρήση της παρακάτω εντολής :

```
gcc  -Wall  -c  main.c
```

Έπειτα για να κάνω σύνδεση του αρχείου main.o που δημιούργησα με την παραπάνω εντολή, με το αρχείο zing.o γράφω την παρακάτω εντολή :

```
gcc  main.o  zing.o  -o  zing
```

Με τον τρόπο αυτό δημιουργώ το εκτελέσιμο αρχείο με όνομα zing, το οποίο αν εκτελεστεί (./zing) εμφανίζει το εξής μήνυμα :

Hello oslabd02!

Παρατήρηση: Το αρχείο zing.h μας χρειάζεται **μόνο** για να μπορέσει να γίνει η μεταγλώττιση του αρχείου main.c και προφανώς δεν παίζει κανένα ρόλο όσο αφορά το linking, αφού εκεί χρειζόμαστε το zing.o .

Ερωτήσεις:

1)Ποιο σκοπό εξυπηρετεί η επικεφαλίδα ;

Η επικεφαλίδα ουσιαστικά περιέχει τη δήλωση του τίτλου μιας ή περισσοτέρων συναρτήσεων. Γενικά η επικεφαλίδα εξυπηρετεί καθώς είναι πολύ πιθανό να επιθυμούμε ένα αρχείο μας να καλεί μια συνάρτηση, η οποία να χρειαστεί να αλλάξει κάποια στιγμή.Συμπεριλαμβάνοντας λοιπόν τη βιβλιοθήκη στον κώδικα του αρχείου μας, έχουμε τη δυνατότητα να βάλουμε τον κώδικα της συνάρτησης μας σε κάποιο άλλο αρχείο, το οποίο θα συνδεθεί αργότερα με τον κώδικα της βασικής συνάρτησης και αυτό μας δίνει την δυνατότητα να προποποιούμε το αρχείο αυτό όποτε εμείς επιθυμούμε.Επίσης μια βιβλιοθήκη μπορεί να φανεί χρήσιμη όταν κάποιες από τις συναρτήσεις μου τις χρησιμοποιώ σε πολλά αρχεία. Στην περίπτωσή μας, η επικεφαλίδα μας είναι αναγκαία διότι 'λέει' στο αρχείο με όνομα main.c που φτιάξαμε, πως η συνάρτηση zing() η οποία καλείται υπάρχει κάπου και για τον λόγο αυτό το compile γίνεται κανονικά και δεν προκύπτει κάποιο error. Αν η επικεφαλίδα έλειπε τότε όταν καλούσαμε τη συνάρτηση zing(), ο compiler δεν θα ήξερε τίποτα σχετικά με αυτή και αυτό θα οδηγούσε σε κάποιο αντοίστιχο μήνυμα λάθους και τελικά στην αποτυχία του compile.

2)Ζητείται κατάλληλο Makefile για τη δημιουργία του εκτελέσιμου της άσκησης.

Για να γίνει κατανοητή η χρήση του Makefile, πριν εκτελέσω την εντολή make διαγράφω το αρχείο αντικειμένων main.o το οποίο προέκυψε από την μεταγλώτιση του αρχείου main.c, όσο και το εκτελέσιμο αρχείο με όνομα zing που προέκυψε μετά το linking, με τις παρακάτω εντολές(ο λόγος για τον οποίο το έκανα θα εξηγηθεί παρακάτω) :

```
rm zing
rm main.o
```

Στη συνέχεια για τη διευκόλυνση της μεταγλώττιση και της σύνδεσης των αρχείων ώστε να προκύψει το εκτελέσιμο αρχείο δημιουργώ ένα Makefile. Το Makefile που υλοποιήθηκε φαίνεται παρακάτω :

```
all: zing

main.o: main.c

	gcc -Wall -c main.c

zing: main.o zing.o

	gcc main.o zing.o -o zing

execute: zing

	./zing
```

Αν μετά τη δημιουργία του Makefile γράψω στο terminal την εντολή make, τότε η εντολή αυτή εντοπίζει το αρχείο Makefile και εκτελεί όλους τους στόχους που χρειάζεται να εκτελέσει ώστε να παράγει το εκτελέσιμο αρχείο (δηλαδή το αρχείο zing).Στην οθόνη μου βλέπω όλα όσα χρειάζεται να γίνουν(μεταγλωττίσεις,συνδέσεις) για να παραχθεί το αρχείο με όνομα zing.Στη συνέχεια αν ξανακαλέσω την εντολή make θα εμφανιστεί ένα μήνυμα το οποίο θα λέει ότι το Makefile είναι ενημερωμέν,δηλαδή ότι όλοι οι στόχοι της ετικέτας all είναι ενημερωμένοι(`make : 'all' is up to date.`) και δεν θα γίνει κανένα από τα προηγούμενα compile και linking. Η μοναδική περίπτωση να εκτελεστεί εκ νέου κάποια από τις παραπάνω εντολές είναι να έχει προκύψει κάποια μεταβολή στα αρχεία που αναφέρονται στις ετικέτες.Επίσης αν αρχικά πατούσα αντι για την εντολή make την εντολή make execute, τότε όχι μόνο να δημιουργούσα το εκτελέσιμο αρχείο(zing) αλλά θα το εκτελούσα επίσης, το οποίο θα είχε ως αποτέλεσμα την τύπωση του μηνύματος: Hello oslabd02! Στο σημείο αυτό θα πρέπει να τονιστεί πως αν δεν έκανα διαγραφή των αρχείων main.o και zing πριν εκτελέσω την εντολή make θα εμφανιζόταν εξ αρχής το μήνυμα το οποίο θα έλεγε πως το Makefile είναι ενημερωμένο,καθώς τα αρχεία zing(εκτελέσιμο) και main.o θα είχαν δημιουργηθεί από το $2^{ο}$ βήμα της άσκησης και δεν θα είχαν διαγραφεί. Το έκανα δηλαδή αυτό για να εξηγήσω πλήρως πως λειτουργεί το Makefile.

3)Γράψτε το δικός σας zing2.o, το οποίο θα περιέχει zing() που θα εμφανίζει διαφορετικό αλλά παρόμοιο μήνυμα με τη zing() του zing.o . Συμβουλευτείται το manual page της getlogin(3). Αλλάξτε το Makefile ώστε να παράγονται 2 εκτελέσιμα, ένα με το zing..o, ένα με το zing2.o, επαναχρησιμοποιώντας το κοινό object file main.o .

Αρχικά θα πρέπει να γράψουμε το ζητούμενο αρχείο zing2.c, το οποίο θα περιέχει παρόμοιο μήνυμα με αυτό που εμφανίζει η zing(). Ο κώδικας αυτού του αρχείου φαίνεται παρακάτω:

```
#include <stdio.h>
#include <unistd.h>

void zing()
{
char *name;
name=getlogin();
if (!name)
    perror("getlogin() error");
else
    printf("This is a new message from %s\n", name);

}
```

Για να μεταγλωττίσω το αρχείο zing2.c χρησιμοποιώ κατά τα γνωστά την εντολή:

```
gcc  -Wall  -c  zing2.c
```

Για να κάνω το linking του zing2.o με το αρχείο main.o χρησιμοποιώ την εντολή:

```
gcc  main.o  zing2.o  -o  out1
```

Με τον τρόπο αυτό δημιουργώ το εκτελέσιμο αρχείο με όνομα out1, το οποίο αν εκτελεστεί (./out1) εμφανίζει το εξής μήνυμα :

This is a new message from oslabd02!

Τέλος, μπορώ να τροποποιήσω το ήδη υπάρχον Makefile ώστε να παράγονται 2 εκτελέσιμα. Το νέο Makefile φαίνεται παρακάτω:

```
all: zing  out1

main.o: main.c
	gcc -Wall -c main.c

zing: main.o zing.o
	gcc main.o zing.o -o zing

zing2.o: zing2.c
	gcc -Wall -c zing2.c

out1: main.o zing2.o
	gcc main.o zing2.o -o out1

executenew: out1
	./out1

execute: zing
	./zing
```

Η λειτουργία του τροποποιημένου Makefile είναι η ίδια με αυτού που εξηγήθηκε παραπάνω, μόνο που το νέο παράγει 2 εκτελέσιμα αρχεία ενώ το παλιό μόνο ένα εκτελέσιμο αρχείο.

4)Έστω ότι έχετε γράψει το πρόγραμμα σας σε ένα αρχείο που περιέχει 500 συναρτήσεις. Αυτή τη στιγμή κάνετε αλλαγές σε μία συνάρτηση. Ο κύκλος εργασίας είναι : αλλαγές στον κώδικα, μεταγλώττιση, εκτέλεση, αλλαγές στον κώδικα, κ.ο.κ. Ο χρόνος μεταγλώττισης είναι μεγάλος, γεγονός που σας καθυστερεί. Πως μπορεί να αντιμετωπισθεί το πρόβλημα αυτό;

Το πρόβλημα αυτό είναι εύκολο να αντιμετωπισθεί αν σκεφτεί κανείς τη χρήση ενός Makefile με ταυτόχρονο διαχωρισμό του αρχείου σε μικρότερα αρχεία. Συγκεκριμένα κάθεμια από τις 500 συναρτήσεις πρέπει να περιέχεται σε κάποιο αρχείο (η καθεμία στο δικό της) και το κύριο μέρος του κώδικα να περιέχεται σε κάποιο άλλο αρχείο. Με τον τρόπο αυτό ανάλογα με τις ετικέτες και τα ορίσματα που θα βάλω στο Makefile θα εκμεταλλευτώ τις ιδιότητες του, τις οποίες ανέφερα παραπάνω, και πλέον όταν τροποποιώ κάποιο-α από τα αρχεία-συναρτήσεις θα εκτελεί μόνο τις εντολές οι οποίες σχετίζονται με τις τροποποιημένες συναρτήσεις, γλιτώνοντας πλέον το compile όλου του κώδικα, γεγονός το οποίο με καθυστερεί πολύ.Δηλαδή πλέον γίνονται μόνο οι απαρραίτητες μεταγλωττίσεις και συνδέσεις , γεγονός το οποίο μου μειώνει σημαντικά το χρόνο μεταγλλώτισης .

5)Ο συνεργάτης σας και εσείς δουλεύατε στο πρόγραμμα foo.c όλη την προηγούμενη βδομάδα. Καθώς κάνατε ένα διάλειμμα και ο συνεργάτης σας δούλευε στον κώδικα, ακούτε μια απελπισμένη κραυγή. Ρωτάτε τι συνέβει και ο συνεργάτης σας λέει ότι το αρχείο foo.c χάθηκε! Κοιτάτε το history του φλοιού και η τελευταία εντολή ήταν η :
gcc –Wall –o foo.c foo.c
Τι συνέβει;

Όταν εκτελέστηκε η παραπάνω εντολή είναι προφανές πως ο χρήστης ήθελε να παράξει το εκτελέσιμο αρχείο, όμως το λάθος που έκανε είναι ότι το αποθήκευσε στο αρχείο foo.c το οποίο είναι το αρχείο κώδικα, δηλαδή έκανε overwrite πάνω του με αποτέλεσμα να χαθεί το αρχείο κώδικα(λόγω του overwrite) και πλέον το αρχείο αυτό να μην είναι προσβάσιμο.

1.2 Συνένωση δύο αρχείων σε τρίτο

Σκοπός αυτής της άσκησης είναι η υλοποίηση ενός προγράμματος το οποίο θα παίρνει δυο αρχεία ως είσοδο και προαιρετικά ένα αρχείο εξόδου και θα τα συνενώνει στο αρχείο εξόδου. Το αρχείο εξόδου είτε θα είναι το τρίτο όρισμα είτε θα είναι το default αρχείο εξόδου της fconc, δηλαδή το fconc.out, αν η συνάρτηση έχει μόνο δυο ορίσματα.
Στο σημείο αυτό θα πρέπει να τονιστεί ότι η άσκηση δεν καθορίζει ποια πρέπει να είναι η συμπεριφορά του προγράμματος μας στην περίπτωση που κληθεί με ορίσματα της μορφής: fconc A B A ή fconc A B B.

Για το λόγο αυτό στη συνέχεια παρατίθενται 2 διαφορετικές εκδοχές. Στη **πρώτη εκδοχή** αν πάρω τέτοιου είδους ορίσματα **εμφανίζω κατάλληλο μήνυμα σφάλματος** και σταματάω την εκτέλεση του προγράμματος και στην **δεύτερη εκδοχή επιτρέπω στα ορίσματα να έχουν αυτή τη μορφή** και πρέπει να παίρνω σωστό αποτέλεσμα στο αρχείο εξόδου.
Οι κώδικες των εκδοχών αυτών φαίνονται παρακάτω :

1η εκδοχή

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>

#define BUFFER_SIZE 512

void create_out_file(char * , char *, char *);
void read_and_write(int,const char *);
int do_read(int, char *, int);
void do_write(int, const char *, int);


int main(int argc, char **argv)
{
        if (argc<3 || argc>=5){
           printf("Usage: ./fconc infile1 infile2 [outfiel]\n");
        }
        else if (argc == 3){
           create_out_file(argv[1], argv[2], NULL);
        }
        else{
           if (strcmp(argv[1], argv[3]) == 0){
                printf("You can't give arguments A B A\n");
           }
           else if(strcmp(argv[2], argv[3]) == 0){
                printf("You can't give arguments A B B\n");
           }
           else create_out_file( argv[1], argv[2] , argv[3]);
        }
        return 0;
}
```

```
void create_out_file(char * file1, char * file2, char * file3)
{
       char * out_file;
       int fd, oflags, mode;
       if (file3 == NULL) out_file = "fconc.out";
       else out_file = file3 ;
       oflags = O_CREAT | O_WRONLY | O_TRUNC;
       mode = S_IRUSR | S_IWUSR;
       fd = open(out_file, oflags, mode);
       if (fd == -1){
            perror("open");
            exit(1);
       }
       else {
            read_and_write(fd, file1);
            read_and_write(fd, file2);
      }
      close(fd);
}

void read_and_write(int fd, const char * file1)
{
      int len,fd1;
      char buffer[BUFFER_SIZE];
      fd1 = open(file1, O_RDONLY);
      if (fd1 == -1){
           perror("open");
           exit(1);
      }
      while ( (len = do_read(fd1, buffer, BUFFER_SIZE)) > 0 ){
           do_write(fd, buffer,len );
      }
      close(fd1);
}


int do_read(int fd,char * buffer, int buf_size)
{
      int result=0, reader;
      while (buf_size > 0){
            reader = read(fd, buffer, buf_size);
            if (reader == -1) {
                 perror("read");
                 exit(1);
            }
            else if (reader == 0)return result;
            else{
                result += reader;
                buf_size -= reader;
                buffer += reader;
            }
       }
       return result;
}
```

```
void do_write(int fd, const char *buffer, int len)
{
      int idx=0;
      int wcnt;
      do{
          wcnt = write(fd,buffer + idx, len - idx);
          if (wcnt == -1){
               perror("write");
               exit(1);
          }
          idx += wcnt;
      }while (idx < len);
}
```

2η εκδοχή

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>

#define BUFFER_SIZE 512

void create_out_file(char * , char *, char *);
void read_and_write(int,const char *);
int do_read(int, char *, int);
void do_write(int, const char *, int);
void create_out_file1(char *, char *);//gia input A B A den stelnw 3o orisma
void create_out_file2(char *, char *);//gia input A B B den stelnw 3o orisma

int main(int argc, char **argv)
{

       if (argc<3 || argc>=5) {
           printf("Usage: ./fconc infile1 infile2 [outfiel]\n");
       }
       else if (argc == 3)  create_out_file(argv[1], argv[2], NULL);
       else{
           if (strcmp(argv[1], argv[3]) == 0){
                 create_out_file1(argv[1], argv[2]);
           }
           else if(strcmp(argv[2], argv[3]) == 0){
                 create_out_file2(argv[1],argv[2]);//gia input A B B
           }
           else create_out_file( argv[1], argv[2] ,argv[3]);//gia input A B A
       }
       return 0;

}
```

```
void create_out_file(char * file1, char * file2, char * file3)
{      //eisodos ths morfhs A B C h A B (OXI A B A H A B B)

       char * out_file;
       if (file3 == NULL) out_file = "fconc.out";
       else out_file = file3 ;
       int oflags = O_CREAT | O_WRONLY | O_TRUNC;
       int mode = S_IRUSR | S_IWUSR;
       int fd = open(out_file, oflags, mode);
       if (fd == -1){
           perror("open");
           exit(1);
       }
       else{
           read_and_write(fd, file1);
           read_and_write(fd, file2);
       }
       close(fd);

}

void create_out_file2(char * file1, char * file3)
{

       //auth h sunarthsh pairnei eisodo ths morfhs A B B kai dinei
       //to swsto apotelesma sto B , dhladh th sunnenosh toy A kai B sto B
       //kai gi auto stelnw mono 2 orismata


       //xrhsimopoiw to fconc.out ws arxeio sto opoio krataw to periexomeno
       //tou arxeiou B kai afou grapsw to A sto B meta grafw to fconc.out sto
       // B kai exw to swsto apotelesma sto B opws epi8umw

       char * out_file="fconc.out";
       int fd, oflags, mode,fd1;
       oflags = O_CREAT | O_WRONLY;
       mode = S_IRUSR | S_IWUSR;
       fd1= open(out_file, oflags,mode);
       fd = open(file3, oflags, mode);
       if (fd == -1 || fd1 == -1){
            perror("open");
            exit(1);
       }
       else{
           //grafw sto fconc.out to B
           read_and_write(fd1,file3);
           //grafw sto B to A
           read_and_write(fd, file1);
           //grafw sto B to fconc.out pou exei to arxiko B
           read_and_write(fd, out_file);
       }
       close(fd);

}
```

```
void create_out_file1(char * file1, char * file2)
{

       //auth h sunarthsh epitrepei eisodo ths morfhs A B A kai dinei
       //to swsto apotelesma sto A , dhladh th sunnenosh toy A kai B sto A
       char * out_file = file1;
       int fd, oflags, mode;
       oflags = O_CREAT | O_WRONLY;
       mode = S_IRUSR | S_IWUSR;
       fd = open(out_file, oflags, mode);
       if (fd == -1){
           perror("open");
           exit(1);
       }
       else{
           read_and_write(fd, file1);
           read_and_write(fd, file2);
       }
       close(fd);

}

void read_and_write(int fd, const char * file1)
{
       int len;
       char buffer[BUFFER_SIZE];
       int fd1 = open(file1, O_RDONLY);
       if (fd1 == -1){
            perror("open");
            exit(1);
       }
       while ( (len = do_read(fd1, buffer, BUFFER_SIZE)) > 0 ){
             do_write(fd, buffer,len );
       }
       close(fd1);
}

int do_read(int fd,char * buffer, int buf_size)
{

       int result=0, reader;
       while (buf_size > 0){
            reader = read(fd, buffer, buf_size);
            if (reader == -1){
                  perror("read");
                  exit(1);
            }
            else if (reader == 0){
                 return result;
            }
            else{
                 result += reader;
                 buf_size -= reader;
                 buffer += reader;
            }
       }
       return result;

}
```

```
void do_write(int fd, const char *buffer, int len)
{
     int idx=0;
     int wcnt;
     do{
        wcnt = write(fd,buffer + idx, len - idx);
        if (wcnt == -1){
             perror("write");
             exit(1);
        }
        idx += wcnt;
     }while (idx < len);
}
```

Μετά τη μεταγλώττιση του προγράμματος fconc.c και για τις 2 εκδοχές είμαι σε θέση πλέον να μπορω να του βάλω σαν είσοδο κάποια αρχεία και να τσεκάρω αν κάνει την συνένωση αυτών σε ένα αρχείο εξόδου.

Έστω ότι τα 2 πρώτα ορίσματα είναι τα αρχεία A,B των οποίων το περιεχόμενο μπορούμε να δούμε πληκτρολογώντας την εντολή : `cat (file name)`

Έστω ότι το περιεχόμενο του αρχείου A είναι το εξής(πατάω cat A) :

```
Hello World from file A
I am a file and nothing more(A)
```

και το περιεχόμενο του αρχείου B είναι το εξής (πατάω cat B):

```
Hello World from file B
I am a file and nothing more(B)
```

Στη συνέχεια γράφω μια σειρά από εντολές και δίπλα το αποτέλεσμα κάθε εκδοχής για να γίνει αντιληπτή η διαφορά τους:

1)fconc A B C : και στις 2 εκδοχές εδώ θα πάρω το ίδιο αποτέλεσμα δηλαδή αν εμφανίσω το αρχείο C με την εντολή cat C θα δω το εξής :

```
Hello World from file A
I am a file and nothing more(A)
Hello World from file B
I am a file and nothing more(B)
```

2)fconc A B : και στις 2 εκδοχές εδώ θα πάρω το ίδιο αποτέλεσμα απλά το αποτέλεσμα της συνένωσης των αρχείων A και B βρίσκεται στο αρχείο εξόδου fconc.out. Άρα αν γράψω την εντολή cat fconc.out παίρνω το εξής αποτέλεσμα στην οθόνη:

```
Hello World from file A
I am a file and nothing more(A)
Hello World from file B
I am a file and nothing more(B)
```

3)fconc A : και στις 2 εκδοχές εδώ θα εμφανιστεί το ίδιο μήνυμα το οποίο έχει να κάνει με το γεγονός ότι έχω μόνο ένα όρισμα το οποίο δεν επιτρέπεται. Το μήνυμα που θα εμφανιστεί στην οθόνη είναι το εξής:

```
Usage: ./fconc infile1 infile2 [outfile]
```

4)fconc A B C D : και στις 2 εκδοχές εδώ θα εμφανιστεί το ίδιο μήνυμα το οποίο έχει να κάνει με το γεγονός ότι έχω πάνω από 3 όρισματα το οποίο δεν επιτρέπεται. Το μήνυμα που θα εμφανιστεί είναι το εξής:

```
Usage: ./fconc infile1 infile2 [outfile]
```

5)fconc A B A : εδώ οι 2 εκδοχές θα έχουν διαφορετικό αποτέλεσμα. Η πρώτη εκδοχή δεν θα αφήσει τον χρήστη να κάνει αυτή τη συνένωση αρχείων και θα του βγάλει το εξής μήνυμα:

```
You can't give arguments A B A
```

Αντίθετα η δεύτερη εκδοχή επιτρέπει τέτοιου είδους συνένωση και πλέον το αρχείο Α έχει την συνένωση του αρχικού αρχείου Α και του αρχείου Β,δηλαδή αν γράψω την εντολή cat A θα δω το εξής στην οθόνη:

```
Hello World from file A
I am a file and nothing more(A)
Hello World from file B
I am a file and nothing more(B)
```

6)fconc A B B: εδώ οι 2 εκδοχές θα έχουν διαφορετικό αποτέλεσμα**.** Η πρώτη εκδοχή δεν θα αφήσει τον χρήστη να κάνει αυτή τη συνένωση αρχείων και θα του βγάλει το εξής μήνυμα:

```
You can't give arguments A B B
```

Αντίθετα η δεύτερη εκδοχή επιτρέπει τέτοιου είδους συνένωση και πλέον το αρχείο Β έχει την συνένωση του αρχείου Α και του αρχικού αρχείου Β,δηλαδή αν γράψω την εντολή cat B θα δω το εξής στην οθόνη:

```
Hello World from file A
I am a file and nothing more(A)
Hello World from file B
I am a file and nothing more(B)
```

Παρατήρηση : Στα παραπάνω παραδείγματα **(5) και (6)** παρατηρώ ότι χάνω το αρχικό περιεχόμενο του Α ή του Β (ανάλογα ποιο παράδειγμα κοιτάω), γεγονός το οποίο αφήνετε στη κρίση του προγραμματιστή να αποφίσει αν θέλει ή όχι να συμβαίνει. Ανάλογα με την απόφαση του, επιλέγει μια από τις παραπάνω υλοποιημένες εκδοχές.

Αν τρέξω το πρόγραμμα fconc.c με χρήση του **strace**, με την εντολή strace ./fconc A B C παίρνω την εξής έξοδο:

```
execve("./exe", ["./exe", "A", "B", "C"], [/* 25 vars */]) = 0

brk(0)                                  = 0x804a000

access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or
directory)

mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb7867000

access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or
directory)

open("/etc/ld.so.cache", O_RDONLY)      = 3

fstat64(3, {st_mode=S_IFREG|0644, st_size=67766, ...}) = 0

mmap2(NULL, 67766, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb7856000

close(3)                                = 0

access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or
directory)

open("/lib/i686/cmov/libc.so.6", O_RDONLY) = 3

read(3,
"\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\260l\1\0004\0\0\0\34"...,
512) = 512

fstat64(3, {st_mode=S_IFREG|0755, st_size=1331684, ...}) = 0

mmap2(NULL, 1337704, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0xb770f000

mmap2(0xb7850000, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x141) = 0xb7850000

mmap2(0xb7853000, 10600, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb7853000

close(3)                                = 0

mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0xb770e000

set_thread_area({entry_number:-1 -> 6, base_addr:0xb770e6c0, limit:1048575,
seg_32bit:1, contents:0, read_exec_only:0, limit_in_pages:1,
seg_not_present:0, useable:1}) = 0

mprotect(0xb7850000, 8192, PROT_READ)   = 0

mprotect(0xb7885000, 4096, PROT_READ)   = 0

munmap(0xb7856000, 67766)               = 0

open("C", O_WRONLY|O_CREAT|O_TRUNC, 0600) = 3

open("A", O_RDONLY)                     = 4

read(4, "Hello World from file A\nI am a fi"..., 512) = 56

read(4, ""..., 456)                     = 0
```

```
write(3, "Hello World from file A\nI am a fi"..., 56) = 56

read(4, ""..., 512)                     = 0

close(4)                                = 0

open("B", O_RDONLY)                     = 4

read(4, "Hello World from file B\nI am a fi"..., 512) = 57

read(4, ""..., 455)                     = 0

write(3, "Hello World from file B\nI am a fi"..., 57) = 57

read(4, ""..., 512)                     = 0

close(4)                                = 0

close(3)                                = 0

exit_group(0)                           = ?
```

Πατατηρώντας την πορεία εξέλιξης του προγράμματος αντιλαμβανόμαστε ότι το system call open() επιστρέφει το αναγνωριστικό του αρχείου, το οποίο λαμβάνει η read() για να διαβάσει. Η read() μας γυρίζει πόσους χαρακτήρες διάβασε και ο αριθμός αυτός εισάγεται στην write() ώστε να γνωρίζει πόσους χαρακτήρες πρέπει να γράψει στο αρχείο εξόδου. Άρα συμπεραίνουμε πως το πρόγραμμά μας λειτουργεί ακρβώς όπως περιμέναμε. Ουσιαστικά η εντολή strace μας επιβεβαίωσε την ορθότητα του προγράμματός μας.