1. Ασκήσεις

1.1. Σύνδεση με αρχείο αντικειμένων

Ο ζητούμενος πηγαίος κώδικας της άσκησης :

```
#include "zing.h"
int main(int argc,char **argv){
    zing();
    return 0 ;
}
```

Με δεδομένα τα object file και header file της συνάρτησης zing() , ακολουθούμε την εξής διαδικασία για την παραγωγή του τελικού εκτελέσιμου αρχείου zing :

Αρχικά , παράγουμε το object file του πηγαίου κώδικα main.c . Σε αυτή τη φάση η κλήση της συνάρτησης zing θα αντικατασταθεί με ένα label : jmp σε μία διεύθυνση μνήμης όπου υπάρχει η υλοποίηση της zing.

```
oslabd01@amorgos:~/Desktop/present/Ex1.1$ gcc -Wall main.c -c -o main.o
```

Στη συνέχεια , παράγουμε το εκτελέσιμο zing συνδέοντας τα δύο object files :

```
oslabd01@amorgos:~/Desktop/present/Ex1.1$ gcc –Wall main.o zing.o -o zing
```

Ακολουθεί η έξοδος εκτέλεσης του προγράμματος zing :

```
oslabd01@amorgos:~/Desktop/present/Ex1.1$ ./zing
Hello oslabd01!
```

Ερώτηση 1 : Ποιο σκοπό εξυπηρετεί η επικεφαλίδα;

Γενικά , τα αρχεία επικεφαλίδων (header files) περιλαμβάνουν μόνο τις δηλώσεις συγκεκριμένων συναρτήσεων και όχι την υλοποίησή τους . Στη συγκεκριμένη άσκηση , στο σώμα εντολών της κύριας συνάρτησης main βρίσκεται και η κλήση της συνάρτησης zing. Ωστόσο , η υλοποίηση της zing δεν βρίσκεται στο πηγαίο κώδικα της main αλλά έχουμε στη διάθεσή μας μόνο το header file και το object file της zing . Συνεπώς , θα πρέπει να συμπεριλάβουμε την επικεφαλίδα της zing στο main.c έτσι ώστε κατά την μετάφραση του πηγαίου κώδικα της main ο compiler να γνωρίζει σε ποιο μέρος θα βρει τη δήλωση της συνάρτησης zing και τι ορίσματα δέχεται (χρησιμοποιείται δηλαδή σαν forward declaration της zing). Έτσι, κατά τη φάση της σύνδεσης των object files main.o και zing.o η υλοποίηση της zing θα «συνδεθεί» με την αντίστοιχη υλοποίηση της main και θα παραχθεί το τελικό εκτελέσιμο αρχείο zing.

Με τη χρήση των επικεφαλίδων και object files των συναρτήσεων γίνεται γνωστή σε άλλους μόνο ο τρόπος κλήσης (ορίσματα) και η λειτουργία της εκάστοτε συνάρτησης χωρίς να παρέχεται σε τρίτους η πραγματική υλοποίησή της (source code).

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

Η διαδικασία μεταγλώττισης και σύνδεσης για την παραγωγή του τελικού εκτελέσιμου zing διευκολύνεται και συστηματοποιείται με την δημιουργία του κατάλληλου Makefile, στο οποίο απεικονίζονται οι εξαρτήσεις μεταξύ των διαφόρων αρχείων που συμμετέχουν στην όλη διαδικασία. Το περιεχόμενο του συγκεκριμένου αρχείου φαίνεται παρακάτω:

```
zing:   zing.o main.o
    gcc –Wall zing.o main.o -o zing

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

Ερώτηση 3 : Γράψτε το δικό σας zing2.o, το οποίο θα περιέχει zing() που θα εμφανίζει διαφορετικό αλλά παρόμοιο μήνυμα με το zing.o. Συμβουλευτείτε το manual page της getlogin(3). Αλλάξτε το Makefile ώστε να παράγονται δύο εκτελέσιμα, επαναχρησιμοποιώντας το κοινό object file.

Ο κώδικας του αρχείου source zing2.c στο οποίο ορίζεται η καινούρια συνάρτηση zing και από το οποίο θα προκύψει με μεταγλώττιση και σύνδεση (με το main.o) το ζητούμενο εκτελέσιμο zing2 φαίνεται παρακάτω:

```c
#include <unistd.h>
#include <sys/types.h>

void zing(){

    char *name ;
    name = getlogin();
    write(1,"Welcome ",8);
    write(1,name,2*sizeof(name));
    write(1,"!\n",2);

}
```

Το παραπάνω source κάνει χρήση της συνάρτησης βιβλιοθήκης getlogin, για την οποία μετά από την κατάλληλη εντολή man διαπιστώσαμε ότι επιστρέφει το username του παρόντος χρήστη του τερματικού με τη μορφή πίνακα χαρακτήρων. Αυτό είναι απαραίτητο να γίνει, καθώς το επιθυμητό μήνυμα για εκτύπωση περιέχει τη συγκεκριμένη πληροφορία. Έτσι, η έξοδος μετά την εκτέλεση του zing2 είναι η ακόλουθη:

```
oslabd01@amorgos:~/Desktop/present/Ex1.1$ ./zing2
Welcome oslabd01!
```

Ας σημειωθεί ότι δημιουργούμε νέο Makefile για τη μεταγλώττιση και σύνδεση τόσο του προγενέστερου εκτελέσιμου, zing, όσο και του νέου, zing2. Η μορφή βέβαια του Makefile τροποποιείται κατάλληλα με τις απαραίτητες προσθήκες. Ενδεικτικά χρησιμοποιείται ως ενδιάμεσος στόχος και για τα δύο εκτελέσιμα το object file main.o και τοποθετείται στην αρχή η ετικέτα all που ακολουθείται από τους δύο τελικούς στόχους, zing και zing2. Όλα αυτά φαίνονται παρακάτω:

```
all:    zing zing2

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

zing2:  zing2.o main.o
    gcc -Wall zing2.o main.o -o zing2

zing2.o:zing2.c
    gcc -Wall -c zing2.c
main.o: main.c
    gcc -Wall -c main.c
clean:
    rm -f main.o zing2.o
```

Ερώτηση 4 : Έστω ότι έχετε γράψει το πρόγραμμά σας σε ένα αρχείο που περιέχει 500 συναρτήσεις. Αυτή τη στιγμή κάνετε αλλαγές μόνο 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. Πώς μπορεί να αντιμετωπισθεί το πρόβλημα αυτό;

Το κλειδί για την οικονομία χρόνου στην περίπτωση αυτή βρίσκεται στον σωστό καταμερισμό του πηγαίου κώδικα. Αναλυτικότερα, δημιουργούμε περισσότερα αρχεία source (εκ των οποίων το ένα με τη main), σε καθένα από τα οποία ενσωματώνουμε ένα υποσύνολο των 500 συναρτήσεων (κατά προτίμηση συσχετιζόμενων μεταξύ τους, έτσι που όταν γίνονται αλλαγές στον κώδικα της μίας να γίνονται συνήθως στο ίδιο στάδιο επεξεργασίας αλλαγές και στις υπόλοιπες). Φυσικά, κάθε αρχείο θα έχει στην αρχή του τις κατάλληλες επικεφαλίδες για τις συναρτήσεις που καλεί και οι οποίες βρίσκονται σε άλλα αρχεία. Γράφουμε το κατάλληλο Makefile με τις εξαρτήσεις και τα προαπαιτούμενα για τη μεταγλώττιση και τη σύνδεση του τελικού εκτελέσιμου. Η χρήση λοιπόν της εντολής make μετά από κάθε κύκλο ανανέωσης (μέρους) του κώδικα θα συνεπάγεται τη μεταγλώττιση μόνο του κώδικα ο οποίος άλλαξε (και όχι όλου όπως προηγουμένως), και επειδή ο χρόνος σύνδεσης των διαφόρων object files που απαιτούνται τώρα είναι μικρός συγκριτικά, η καθυστέρηση που εισάγεται στον κύκλο εργασίας μειώνεται σημαντικά σε σχέση με την πολιτική του ενός source αρχείου.

Ερώτηση 5 : Ο συνεργάτης σας και εσείς δουλεύατε στο πρόγραμμα foo.c όλη την προηγούμενη εβδομάδα. Καθώς κάνατε ένα διάλειμμα και ο συνεργάτης σας δούλευε στον κώδικα, ακούτε μια απελπισμένη κραυγή. Ρωτάτε τι συνέβη 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 συνέβη;

Η εντολή που πληκτρολογήθηκε είχε σκοπό να μεταγλωττίσει και να συνδέσει το αρχείο source foo.c για να δημιουργηθεί το αντίστοιχο εκτελέσιμο. Όμως, έχει λανθασμένα

χρησιμοποιηθεί ως όνομα για το εκτελέσιμο αυτό το ίδιο όνομα με αυτό του source αρχείου, δηλαδή foo.c, με αποτέλεσμα η μεταγλώττιση και η σύνδεση να γίνουν κανονικά, απλώς το foo.c να περιέχει πλέον γλώσσα μηχανής και να μην είναι επεξεργάσιμο στο μέλλον και ουσιαστικά να χάσουμε τον πηγαίο κώδικα της foo. Το συμπέρασμα είναι ότι οι εντολές gcc καλό είναι να περιέχονται στο Makefile, ώστε να συστηματοποιείται η διαδικασία μεταγλώττισης και σύνδεσης, και όχι να γράφονται κάθε φορά στη γραμμή εντολών, οπότε και αυξάνεται η πιθανότητα να γίνει έστω μία φορά ανάλογο λάθος.

1.2. Συνένωση δύο αρχείων σε τρίτο

Για την υλοποίηση του ζητούμενου προγράμματος χρησιμοποιούνται οι παρακάτω βοηθητικές συναρτήσεις :

- **checkArgs** : η συνάρτηση ελέγχει αν ο χρήστης έδωσε σωστό αριθμό ορισμάτων και σε περίπτωση ορθής εισαγωγής , αποθηκεύει τα ονόματα των αρχείων εισόδου και εξόδου (προεπιλεγμένο ή επιλογής του χρήστη) σε κατάλληλες μεταβλητές/θέσεις πίνακα για να χρησιμοποιηθούν από τη συνάρτηση κυρίου προγράμματος και άλλες βοηθητικές συναρτήσεις . Σε περίπτωση λάθους , απλά εμφανίζει μήνυμα βοήθειας που επεξηγεί τη σωστή σύνταξη των ορισμάτων του προγράμματος.

- **manageOpen** : η συνάρτηση δέχεται ως ορίσματα το όνομα ενός αρχείου και τις κατάλληλες σημαίες για ανάγνωση/εγγραφή (O_RDONLY,O_WRONLY,O_CREAT και O_TRUNC) . Χρησιμοποιεί την κλήση συστήματος της open μαζί με το κατάλληλο mode για ανάγνωση ή εγγραφή και επιστρέφει έναν ακέραιο (file descriptor) που λειτουργεί ως αναγνωριστικό για τις υπόλοιπες κλήσεις συστήματος . Σε περίπτωση σφάλματος , εμφανίζει κατάλληλο μήνυμα και επιστρέφει την τιμή -1.

- **doWrite** : η συνάρτηση δέχεται ως ορίσματα τον file descriptor ενός αρχείου προς εγγραφή, έναν πίνακα με χαρακτήρες προς εγγραφή και το μέγεθος του πίνακα σε byte. Χρησιμοποιεί επαναληπτικά τη κλήση συστήματος της write μέχρις ότου γραφούν όλοι οι χαρακτήρες στο αρχείο που υποδεικνύει ο file descriptor . Σε περίπτωση σφάλματος , εμφανίζει κατάλληλο μήνυμα και τερματίζει την εκτέλεση του προγράμματος.

- **write_file** : η συνάρτηση δέχεται ως ορίσματα τον file descriptor του αρχείου εξόδου και το όνομα ενός αρχείου εισόδου . Μέσω της συνάρτησης manageOpen υπολογίζει τον file descriptor για το αρχείο εισόδου και στη συνέχεια χρησιμοποιεί επαναληπτικά τη κλήση συστήματος της read μέχρις ότου διαβαστούν όλοι οι χαρακτήρες από το αρχείο εισόδου . Σε κάθε κλήση της read , το αποτέλεσμα ανάγνωσης αποθηκεύεται σε ένα buffer χωρητικότητας 1024 bytes και καλείται η συνάρτηση doWrite έτσι ώστε να γραφούν στο αρχείο εξόδου οι χαρακτήρες που μόλις διαβάστηκαν .Τέλος, κλείνει το αρχείο εισόδου μέσω της κλήσης συστήματος της close . Σε περίπτωση σφάλματος , εμφανίζει κατάλληλο μήνυμα και τερματίζει την εκτέλεση του προγράμματος.

Η συνένωση των δύο αρχείων εισόδου σε ένα αρχείο εξόδου πραγματοποιείται από την κύρια συνάρτηση του προγράμματος (main) . Αρχικά , αφού ελέγξουμε ότι ο χρήστης έδωσε τον σωστό αριθμό ορισμάτων μέσω της checkArgs , ανοίγουμε το αρχείο εξόδου (αν υπάρχει το μηδενίζουμε ενώ αν δεν υπάρχει το δημιουργούμε) μέσω της manageOpen και στη συνέχεια αντιγράφουμε τα αρχεία εισόδου στο αρχείο εξόδου μέσω της write_file.

Στη συγκεκριμένη υλοποίηση του προγράμματος συνένωσης δύο αρχείων εισόδου λαμβάνουμε υπόψη και την περίπτωση όπου ο χρήστης εισάγει ως αρχείο εξόδου ένα από τα αρχεία εισόδου . Η περίπτωση αυτή «ανιχνεύεται» από τη συνάρτηση κυρίου προγράμματος και τότε δημιουργείται ένα προσωρινό αρχείο (tmp) μέσω της **mkstemp** (η οποία δημιουργεί το αρχείο και επιστρέφει τον file descriptor του). Στη συνέχεια , αντιγράφονται τα περιεχόμενα του αρχείου εισόδου-εξόδου στο προσωρινό αρχείο. Κατά τη συνένωση των αρχείων εισόδου στο αρχείο εξόδου , μόλις αναγνωριστεί ότι το αρχείο εισόδου είναι και το αρχείο εξόδου , η συνάρτηση write_file δεν καλείται με το όνομα του αρχείου εισόδου αλλά με εκείνο του προσωρινού αρχείου. Τέλος , το προσωρινό αρχείο διαγράφεται κατά το τέλος εκτέλεσης του προγράμματος μέσω της κλήσης συστήματος της **unlink** .

Σημείωση : στην περίπτωση των δύο αρχείων εισόδου οι συναρτήσεις checkArgs και manageOpen θα μπορούσαν να αντικατασταθούν από απλά if-then statements .Ωστόσο, χάρη σε αυτές το πρόγραμμα μπορεί εύκολα να επεκταθεί και για την περίπτωση άπειρων αρχείων εισόδου (Προαιρετική άσκηση 3) .

Παρατίθεται ο πηγαίος κώδικας (source code) της άσκησης :

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

char tmp[20] = "";
int manageOpen(const char * , int); //fowarding

void doWrite(int fd , const char *buff , int len){
    int size,written;
    written = 0;
    size = len;
    while (written = write(fd,buff,size))
    {   size -=written;
        if (written ==-1){
            perror("Error while writing file");
            exit(1);
        }
    }
}
```

```c
// function called to write an infile to the output file using a
// buffer and repeatedly calling function doWrite until all bytes
// are successfully written to output file.

void write_file(int fd , const char *infile){
    int size,infd ;
    char buffer[1024];
    if ((infd = manageOpen(infile,O_RDONLY)) == -1) {
        exit(1);
    }
    while ((size = read(infd,buffer,1024))){
        doWrite(fd,buffer,size);
    }
    if (close(infd) < 0 ) {
        perror("Error while closing input File");
        exit(1);
    }
}
//function called to check if arguments were inserted correctly by
//the use and stores infiles/outfile
//(or creates the default) . In case of error , function just
//displays  a help message !
int checkArgs(int argc , char **argv , char ***infile , char **
outfile){

    int i,fin;
    if (argc < 3 ) {
        printf("Usage: ./fconc infile1 infile2 [outfile
                                        (default:fconc.out)]\n");
        return -1 ;
    }
    else {
        if (argc < 4) {
            fin = argc -1 ;
            *outfile = "fconc.out";
        }
        else {
            fin = argc -2 ;
            *outfile = argv[argc-1];
        }
        *infile = (char **)malloc((fin)*sizeof(char*));
        for (i = 0 ; i < fin; i++)
            (*infile)[i] = argv[i+1];
    }
    return 0 ;
}

// function called for read/write using name and the correct flags
// returns the (int) file descriptor (as return by open) .In case of
// error displays error message and returns -1
int manageOpen(const char *file , int oflags){

    int fd ;
    fd = open(file,oflags,S_IRUSR|S_IWUSR);
    if (fd == -1){
        perror(file);
        return -1 ;
    }
    return fd;

}
```

```c
int main (int argc , char **argv)
{
    char **infile,*outfile;
    int outfd,tmpfd;
    int i ,fin;
    if (checkArgs(argc,argv,&infile,&outfile) == -1 )
        return -1 ;
    if (argc < 4) fin = argc -1 ;
    else fin = argc -2 ;
    i = 0 ;
    while ((i<fin) && (strcmp(tmp,"") == 0)){
      //search for case one infile == outfile
        if (strcmp(infile[i],outfile) ==0){
            //Create a temporary file to copy the initial context of
            //the infile that will be also the outfile
            strncpy(tmp,"./temp-XXXXXX",sizeof tmp);
            if ((tmpfd = mkstemp(tmp)) == -1) {
                perror("Problem while creating tmp file");
                exit(1);
            }
            //copy the initial state of the future outfile to temp
            //file !
            write_file(tmpfd,infile[i]);
        }
    i++;
    }
    if ((outfd = manageOpen(outfile,O_WRONLY|O_CREAT|O_TRUNC)) ==-1)
      return -1 ;

    for (i = 0 ; i < fin ; i++){
        if (strcmp(infile[i],outfile) ==0)
            // bypass the infile using tmp
            write_file(outfd,tmp);
        else // infile is different from outfile
            write_file(outfd,infile[i]);
    }

    if (close(outfd) < 0 ) {
        perror("Error while closing output file");
        exit(1);
    }
    free(infile);
    unlink(tmp);
    return 0;
}
```

Ακολουθούν παραδείγματα εκτέλεσης του προγράμματος :

```
oslabd01@amorgos:~/Desktop/present/Ex1.2$ ./fconc A
Usage: ./fconc infile1 infile2 [outfile (default:fconc.out)]
oslabd01@amorgos:~/Desktop/present/Ex1.2$ ./fconc A B
A: No such file or directory
oslabd01@amorgos:~/Desktop/present/Ex1.2$ echo 'So long,' > A
oslabd01@amorgos:~/Desktop/present/Ex1.2$ echo 'and thanks for
all the fish!' > B
oslabd01@amorgos:~/Desktop/present/Ex1.2$ ./fconc A B
```

```
oslabd01@amorgos:~/Desktop/present/Ex1.2$ cat fconc.out
So long,
and thanks for all the fish!
oslabd01@amorgos:~/Desktop/present/Ex1.2$ ./fconc A B C
oslabd01@amorgos:~/Desktop/present/Ex1.2$ cat C
So long,
and thanks for all the fish!
```

Ερώτηση 1 : Εκτελέστε ένα παράδειγμα του fconc χρησιμοποιώντας την εντολή strace. Αντιγράψτε το κομμάτι της εξόδου της strace που προκύπτει από τον κώδικα που γράψατε.

Η ζητούμενη έξοδος της strace :

```
oslabd01@amorgos:~/Desktop/present/Ex1.2$ strace ./fconc A B C
execve("./fconc", ["./fconc", "A", "B", "C"], [/* 16 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) = 0xb786e000
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=63569, ...}) = 0
mmap2(NULL, 63569, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb785e000
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\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) = 0xb7717000
mmap2(0xb7858000, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x141) = 0xb7858000
mmap2(0xb785b000, 10600, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb785b000
close(3)                                = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -
1, 0) = 0xb7716000
set_thread_area({entry_number:-1 -> 6, base_addr:0xb77166c0,
limit:1048575, seg_32bit:1, contents:0, read_exec_only:0,
limit_in_pages:1, seg_not_present:0, useable:1}) = 0
mprotect(0xb7858000, 8192, PROT_READ)   = 0
mprotect(0xb788c000, 4096, PROT_READ)   = 0
munmap(0xb785e000, 63569)               = 0
brk(0)                                  = 0x804a000
brk(0x806b000)                          = 0x806b000
open("C", O_WRONLY|O_CREAT|O_TRUNC, 0600) = 3
open("A", O_RDONLY)                     = 4
read(4, "So long,\n"..., 1024)          = 9
write(3, "So long,\n"..., 9)            = 9
write(3, ""..., 0)                      = 0
read(4, ""..., 1024)                    = 0
close(4)                                = 0
```

```
open("B", O_RDONLY)                     = 4
read(4, "and thanks for all the fish!\n"..., 1024) = 29
write(3, "and thanks for all the fish!\n"..., 29) = 29
write(3, ""..., 0)                      = 0
read(4, ""..., 1024)                    = 0
close(4)                                = 0
close(3)                                = 0
unlink("")                              = -1 ENOENT (No such file or
directory)
exit_group(0)                           = ?
```

Μέσω της εντολής strace μπορούμε να παρακολουθήσουμε όλες τις κλήσεις συστήματος που χρησιμοποιούνται από ένα πρόγραμμα καθώς επίσης και τις επιστροφές-σήματα που δέχεται από αυτές . Στη συγκεκριμένη εκτέλεση του προγράμματος fconc με ορίσματα A B (υπάρχοντα αρχεία εισόδου) και C (προς δημιουργία αρχείο εξόδου) παρατηρούμε στο κομμάτι της εξόδου που βρίσκεται σε πλαίσιο πως αρχικά το πρόγραμμα «ανοίγει» το αρχείο C (ουσιαστικά το δημιουργεί μέσω του flag O_CREAT με file descriptor = 3) , ανοίγει για ανάγνωση το αρχείο A (file descriptor = 4) και στη συνέχεια διαβάζει από το αρχείο A και γράφει στο αρχείο C μέχρις ότου να μη διαβαστεί κανένα byte από το αρχείο A (οι read/write επιστρέφουν το πλήθος των bytes που διάβασαν/έγραψαν με επιτυχία). Στη συνέχεια , κλείνει το αρχείο A και επαναλαμβάνεται η παραπάνω διαδικασία ανάγνωσης – εγγραφής για τα αρχεία B και C . Τέλος , το πρόγραμμα κλείνει τα αρχεία και επειδή δεν έχει δημιουργεί προσωρινό αρχείο η κλήση συστήματος της unlink επιστρέφει -1 .

Προαιρετικές Ασκήσεις

Ερώτηση 1 : Χρησιμοποιήστε την εντολή strace για να εντοπίσετε με ποια κλήση συστήματος υλοποιείται η εντολή strace. Υπόδειξη: με strace -o file η έξοδος της εντολής τοποθετείται στο αρχείο file.

Παρατίθεται ένα μέρος της εξόδου της εκτέλεσης : strace strace ./zing

```
trace(PTRACE_PEEKDATA, 4782, 0xb770e004, [0x736f206f]) = 0
ptrace(PTRACE_PEEKDATA, 4782, 0xb770e008, [0x6462616c]) = 0
ptrace(PTRACE_PEEKDATA, 4782, 0xb770e00c, [0xa213130]) = 0
write(2, "write(1, \"Hello oslabd01!\\n\"..., "..., 35write(1, "Hello
oslabd01!\n"..., 16) = 35
ptrace(PTRACE_SYSCALL, 4782, 0x1, SIG_0Hello oslabd01!
) = 0
--- SIGCHLD (Child exited) @ 0 (0) ---
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
wait4(-1, [{WIFSTOPPED(s) && WSTOPSIG(s) == SIGTRAP}], __WALL, NULL)
= 4782
rt_sigprocmask(SIG_BLOCK, [HUP INT QUIT PIPE TERM], NULL, 8) = 0
ptrace(PTRACE_PEEKUSER, 4782, 4*ORIG_EAX, [0x4]) = 0
ptrace(PTRACE_PEEKUSER, 4782, 4*EAX, [0x10]) = 0
write(2, ")    = 16\n"..., 10)    = 16
)          = 10
ptrace(PTRACE_SYSCALL, 4782, 0x1, SIG_0) = 0
```

Παρατηρούμε πως η εντολή strace υλοποιείται με την κλήση συστήματος **ptrace**. Ανατρέχοντας στο man 2 ptrace βρίσκουμε πως η εντολή ptrace χρησιμοποιείται για την «ιχνηλασία» (tracing) άλλων διαδικασιών . Κατά την κλήση συστήματος της ptrace , η κύρια διαδικασία μπορεί να παρατηρεί και να ελέγχει τη διαδικασία εκτέλεσης μίας άλλης διαδικασίας καθώς επίσης να εξετάζει και να αλλάζει την «εικόνα πυρήνα» (core image) και τους καταχωρητές της διαδικασίας.

Ερώτηση 2 : Χρησιμοποιώντας το πρόγραμμα gdb (GNU debugger), μπορείτε να δείτε την assembly συναρτήσεων. Παρατηρήστε ότι εκτός των απολύτων διευθύνσεων, η μοναδική αλλαγή στην assembly είναι το όρισμα της εντολής call (κλήση συνάρτησης). Πού οφείλεται η αλλαγή; Ποιος την έκανε;

Οι έξοδοι του προγράμματος gdb είναι ακριβώς ίδιες με εκείνες της εκφώνησης και παραλείπονται για λόγους συντομίας .

Η μοναδική αλλαγή που παρατηρείται στις δύο assemblies, στο όρισμα της εντολής call, οφείλεται στο γεγονός ότι στη μεν πρώτη περίπτωση έχουμε κάνει disassemble το **object file** main.o, το οποίο δεν έχει υποστεί ακόμη τη διαδικασία σύνδεσης, στη δε δεύτερη περίπτωση το disassembling εφαρμόστηκε στο **εκτελέσιμο αρχείο** zing, το οποίο είναι προϊόν της **σύνδεσης** του αρχείου main.o με το zing.o. Άρα, ο **linker** κατά τη διαδικασία της σύνδεσης των δύο αρχείων αντικειμένων βρήκε την πραγματική διεύθυνση στην οποία πρέπει να πάει ο program counter αμέσως μετά την κλήση της συνάρτησης zing (που σχετίζεται με τη θέση του zing.o) και την τοποθέτησε στη θέση αυτής που υπήρχε προηγουμένως στο main.o, καθώς αυτή είναι που θα χρειαστεί να αποτελεί το όρισμα της call κατά την εκτέλεση του προγράμματος για την επιτυχή έκβασή της.

Ερώτηση 3 : Γράψτε το πρόγραμμα της άσκησης 1.2, ώστε να υποστηρίζει αόριστο αριθμό αρχείων εισόδου (π.χ. 1, 2, 3, 4, ...). Θεωρήστε ότι το τελευταίο όρισμα είναι πάντα το αρχείο εξόδου.

Για την υλοποίηση του ζητούμενου προγράμματος χρησιμοποιούνται εκτός από τις βοηθητικές συναρτήσεις της άσκησης 1.2 και οι παρακάτω συναρτήσεις :

- **searchIO** : η συνάρτηση ελέγχει αν ο χρήστης έδωσε ως αρχείο εξόδου ένα από τα αρχεία εισόδου και επιστρέφει την θέση (index) που βρέθηκε για πρώτη φορά στον πίνακα infile. Αν το αρχείο εξόδου δεν είναι κάποιο από τα αρχεία εξόδου τότε επιστρέφει την τιμή -1.

- **tester** : η συνάρτηση καλείται από την συνάρτηση write_file στην περίπτωση όπου παρουσιάστηκε σφάλμα κατά την κλήση συστήματος της open για «άνοιγμα» ενός αρχείου εξόδου προς ανάγνωση . Αν έχει δημιουργηθεί ένα προσωρινό αρχείο (δηλαδή το αρχείο εξόδου είναι κάποιο από τα αρχεία εισόδου) τότε επαναφέρει το συγκεκριμένο αρχείο εξόδου στην αρχική του κατάσταση και διαγράφει το προσωρινό αρχείο . Αν το αρχείο εξόδου δεν είναι και αρχείο εισόδου τότε απλά διαγράφει το αρχείο εξόδου . Σε κάθε περίπτωση, εμφανίζεται κατάλληλο επεξηγηματικό μήνυμα λάθους και ο έλεγχος επιστρέφει στην συνάρτηση write_file.

Σημείωση : Η συγκεκριμένη υλοποίηση της συνένωσης αρχείων διαφέρει σε σχέση με τη συμπεριφορά της εντολής cat . Σε περίπτωση λανθασμένης εισαγωγής ενός αρχείου εισόδου (π.χ. δεν υπάρχει) , το πρόγραμμα ενημερώνει το χρήστη και δεν δημιουργεί το αρχείο εξόδου . Με τον τρόπο αυτό διασφαλίζεται ότι το αρχείο εξόδου θα είναι η συνένωση όλων των αρχείων εισόδου και αποφεύγεται πιθανή αλλοίωση ενός αρχείου εισόδου (αν π.χ. το αρχείο εισόδου είναι και αρχείο εξόδου) . Αντίθετα , η εντολή cat – σε περίπτωση λάθους – επιστρέφει το αποτέλεσμα συνένωσης των αρχείων ακριβώς πριν από το λάθος .

Παρατίθεται ο πηγαίος κώδικας (source code) της άσκησης :

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

char tmp[20] = "";
char *outfile;
//foward definitions
void tester();
int manageOpen(const char*, int);

void doWrite(int fd , const char *buff , int len){
    write(fd,buff,len);
}

// function called to write an infile to the output file using a
// buffer and repeatedly calling function doWrite until all bytes
// are successfully written to output file.in case of error when
// opening an infile , function calls tester to handle the error
// (see void tester for details)

void write_file(int fd , const char *infile){
    int size,infd ;
    char buffer[1024];
    if ((infd = manageOpen(infile,O_RDONLY)) == -1) {
        tester();
        exit(1);
    }
    while ((size = read(infd,buffer,1024))){
        doWrite(fd,buffer,size);
    }
    if (close(infd) < 0 ) {
        perror("Error while closing input File");
        exit(1);
    }
}

// function is called when an infile cannot be opened correctly in order
// to test if a tmp was previously created (i.e input file is also the
// output file and restores the output file to its initial state
// if a tmp does not exist, function just deletes the outfile
```

```c
void tester() {
    int outfd;
    if (strcmp(tmp,"")!=0)
        {
        if ((outfd = manageOpen(outfile,O_WRONLY|O_TRUNC)) ==-1)
      return;
        write_file(outfd,tmp);
        printf("Restoring file %s to its initial state!\n",outfile);
        }
    else {
        if (remove(outfile) != 0)
            perror("Unable to delete outfile\n");
    }

}
// function called to check if arguments were inserted correctly by the
// user and stores infiles/outfile (or creates the default). In case of
// error, function just displays a help message !
int checkArgs(int argc , char **argv , char ***infile , char **
outfile){

    int i,fin;
    if (argc < 3 ) {
        printf("Usage: ./fconc infile1 infile2 (...)[outfile
                                            (default:fconc.out)]\n");
        return -1 ;
    }
    else {
        if (argc < 4) {
            fin = argc -1 ;
            *outfile = "fconc.out";
        }
        else {
            fin = argc -2 ;
            *outfile = argv[argc-1];
        }
        *infile = (char **)malloc((fin)*sizeof(char*));
            for (i = 0 ; i < fin; i++)
                    (*infile)[i] = argv[i+1];
    }
    return 0 ;
}

// function called for read/write using filename and the correct flags
// returns the (int) file descriptor (as return by open) .In case of
// error displays error message and returns -1
int manageOpen(const char *file , int oflags){
    int fd ;
        fd = open(file,oflags,S_IRUSR|S_IWUSR);
    if (fd == -1){
        perror(file);
        return -1 ;
    }
    return fd;

}

// function called only when the output file is user defined
// in order to search if an input file is also the outfile and
// returns its position at infile array
int searchIO(int argc, char **infile , char *outfile){
```

```
    int found = -1 ;
    int i = 0 ;
    while ((found == -1)&&(i < argc -2)){
        if (strcmp(infile[i],outfile) ==0)
            found = i ;
        i++;
    }

    return found ;
}

int main (int argc , char **argv)
{
    char **infile;
    int outfd,position,tmpfd;
    int i ,fin;
    if (checkArgs(argc,argv,&infile,&outfile) == -1 )
        return -1 ;

    if ((position = searchIO(argc,infile,outfile)) != -1)
      // an input file is also the output file
    {
    //Create a temporary file to copy the initial context of the
      //infile that will be also the outfile
        strncpy(tmp,"./temp-XXXXXX",sizeof tmp);
        if ((tmpfd = mkstemp(tmp)) == -1) {
            perror("Problem while creating tmp file");
            exit(1);
        }
        write_file(tmpfd,infile[position]);
        if ((outfd = manageOpen(outfile,O_WRONLY|O_CREAT|O_TRUNC)) ==-1)
            return -1 ;
        for (i = 0 ; i < argc -2 ; i++){
            if (strcmp(infile[i],outfile) ==0)
                write_file(outfd,tmp);
            else // infile is different from outfile
                write_file(outfd,infile[i]);
        }
    }
    else {
        if ((outfd = manageOpen(outfile,O_WRONLY|O_CREAT|O_TRUNC)) ==-1)
            return -1 ;
        if (argc < 4) fin = argc -1 ;
        else fin = argc -2 ;
        for (i = 0 ; i <  fin; i++)
            {
                write_file(outfd,infile[i]);
            }
    }
    if (close(outfd) < 0 ) {
        perror("Error while closing output file");
        exit(1);
    }
    unlink(tmp);
    free(infile);
    return 0 ;
}
```

Ακολουθούν παραδείγματα εκτέλεσης του προγράμματος :

```
oslabd01@serifos:~/Desktop/present/Ex1.2$ ./bonus
Usage: ./fconc infile1 infile2 (...) [outfile
(default:fconc.out)]
oslabd01@serifos:~/Desktop/present/Ex1.2$ ./bonus A
Usage: ./fconc infile1 infile2 (...) [outfile
(default:fconc.out)]
oslabd01@serifos:~/Desktop/present/Ex1.2$ ./bonus A B
A: No such file or directory
oslabd01@serifos:~/Desktop/present/Ex1.2$ echo 'Goodbye,' > A
oslabd01@serifos:~/Desktop/present/Ex1.2$ echo 'and thanks for
all the fish!' > B
oslabd01@serifos:~/Desktop/present/Ex1.2$ ./bonus A B
oslabd01@serifos:~/Desktop/present/Ex1.2$ cat fconc.out
Goodbye,
and thanks for all the fish!
oslabd01@serifos:~/Desktop/present/Ex1.2$ ./bonus A B C
oslabd01@serifos:~/Desktop/present/Ex1.2$ cat C
Goodbye,
and thanks for all the fish!
oslabd01@serifos:~/Desktop/present/Ex1.2$ echo 'six times nine
is,of course, fifty-four' > D
oslabd01@serifos:~/Desktop/present/Ex1.2$ echo 'six times seven
is forty two' > E
oslabd01@serifos:~/Desktop/present/Ex1.2$ ./bonus A B D E C
oslabd01@serifos:~/Desktop/present/Ex1.2$ cat C
Goodbye,
and thanks for all the fish!
six times nine is,of course, fifty-four
six times seven is forty two
oslabd01@serifos:~/Desktop/present/Ex1.2$ ./bonus A B D E C A
Input file is output file!
oslabd01@serifos:~/Desktop/present/Ex1.2$ cat A
Goodbye,
and thanks for all the fish!
six times nine is,of course, fifty-four
six times seven is forty two
Goodbye,
and thanks for all the fish!
six times nine is,of course, fifty-four
six times seven is forty two
oslabd01@serifos:~/Desktop/present/Ex1.2$ ./bonus A B D E C A V G
V: No such file or directory
oslabd01@serifos:~/Desktop/present/Ex1.2$ cat G
cat: G: No such file or directory
oslabd01@serifos:~/Desktop/present/Ex1.2$ ./bonus A B D E C A V A
Input file is output file!
V: No such file or directory
Restoring file A to its initial state!
oslabd01@serifos:~/Desktop/present/Ex1.2$ cat A
Goodbye,
and thanks for all the fish!
six times nine is,of course, fifty-four
six times seven is forty two
```

```
Goodbye,
and thanks for all the fish!
six times nine is,of course, fifty-four
six times seven is forty two
```

Ερώτηση 4 : Εάν τρέξετε το εκτελέσιμο /home/oslab/code/whoops/whoops θα δώσει:

```
$ /home/oslab/code/whoops/whoops
Problem!
```

Θεωρήστε ότι πραγματικά υπάρχει πρόβλημα. Εντοπίστε το.

Τρέχοντας το εκτελέσιμο της εκφώνησης πράγματι προκύπτει :

```
oslabd01@amorgos:/home/oslab/code/whoops$ ./whoops
Problem!
```

Για να εντοπίσουμε σε ποιο σημείο βρίσκεται το πραγματικό πρόβλημα , χρησιμοποιούμε την εντολή strace έτσι ώστε να παρακολουθήσουμε όλες τις κλήσεις συστήματος του προγράμματος και τα πιθανά σφάλματα :

```
oslabd01@amorgos:/home/oslab/code/whoops$ strace ./whoops
execve("./whoops", ["./whoops"], [/* 16 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) = 0xb77bd000
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=63569, ...}) = 0
mmap2(NULL, 63569, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb77ad000
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) = 0xb7666000
mmap2(0xb77a7000, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x141) = 0xb77a7000
mmap2(0xb77aa000, 10600, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb77aa000
close(3)                                = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0xb7665000
set_thread_area({entry_number:-1 -> 6, base_addr:0xb76656c0,
limit:1048575, seg_32bit:1, contents:0, read_exec_only:0,
limit_in_pages:1, seg_not_present:0, useable:1}) = 0
mprotect(0xb77a7000, 8192, PROT_READ)   = 0
mprotect(0xb77db000, 4096, PROT_READ)   = 0
munmap(0xb77ad000, 63569)               = 0
open("/etc/shadow", O_RDONLY)           = -1 EACCES (Permission denied)
write(2, "Problem!\n"..., 9Problem!
)           = 9
```

Από την έξοδο της strace παρατηρούμε πως κατά τη διάρκεια εκτέλεσης το πρόγραμμα προσπαθεί να «ανοίξει» προς ανάγνωση το αρχείο με όνομα shadow χωρίς να διαθέτει τα κατάλληλα δικαιώματα με αποτέλεσμα το system call της open να επιστρέφει την τιμή -1 .