



ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΜΑΤΙΚΗΣ

Άρτεμις Στεφανίδου

2^η Εργασία στο μάθημα **Λειτουργικά Συστήματα**

Ταύρος, 26 Ιανουαρίου 2021

Περιεχόμενα

Άσκηση 2	2
Κώδικας	2
Τρόπος Εκτέλεσης	3
Ενδεικτικές εκτελέσεις (screenshots):	3
Βασική εκτέλεση του προγράμματος	3
Παρατηρήσεις/σχόλια	3
Δημιουργία νέων διεργασιών και φυσιολογικός τερματισμός τους	3
Παρατηρήσεις/σχόλια	4
Δημιουργία νημάτων και φυσιολογικός τερματισμός τους	4
Παρατηρήσεις/σχόλια	5
Δημιουργία νημάτων και φυσιολογικός τερματισμός τους	5
Παρατηρήσεις/σχόλια	6
Γενικά Σχόλια/Παρατηρήσεις	6
Με δυσκόλεψε / δεν υλοποίησα	6
Συνοπτικός Πίνακας	7

Άσκηση 2

Κώδικας :

Ο κώδικας της 2ης εργασίας που δημιουργήθηκε μαζί με τα σχόλια είναι:

```
it21996.c

#include "wordcounter.h"
#include <signal.h>
```

```

#include <dirent.h>
#include <sys/types.h>
#include <sys/wait.h>

//Display a message to the user and will continue to run
void sig_Handler(int sig_num)
{
    printf("Ignoring signal: %d", sig_num);
}

//Check if a file-char array contains a not ASCII character
int isnot_ascii(int filesize,char *buffer)
{
    int i;
    for(i=0;i<filesize;i++)
    {
        //ASCII has from 0 to 127
        if((int)buffer[i]>127 || (int)buffer[i]<0 ){
            printf("\tThe file is not in ASCII\n");
            return 1;//true
        }
    }
    return 0;//false
}

int main(int argc, char *argv[])
{
    DIR *pDir;
    DIR *not_file;
    struct dirent *de;
    char* filename;
    size_t filename_l;
    pid_t pid;
    int count_for_fork=0;

    //Check if user gives correct arguments from terminal
    if(argc>2)
    {
        printf("\tOptionally you have to give one directory.If not given
will be listed in the current directory\n");
        exit(1);
    }
}

```

```

    if(argc == 1)
    {
        //Get current directory if user don't give a second argument
        from terminal
        argv[1]=".";

    }

    //open the directory given from user
    pDir = opendir (argv[1]);

    //Check if the directory opens
    if (pDir == NULL)
    {
        //Print to user the corresponding message
        perror("");

        //exit from the program
        exit(1);

    }

    int fd, characters;

    //read one per one the contents of the folder
    while ((de = readdir(pDir)) != NULL)
    {
        //get the name of the file that read
        filename=de->d_name;

        //calculate the size of the path for the file that read
        int filename_size=strlen(argv[1])+1+strlen(filename);

        //create a char array for file path with the correct size
        char buffer_of_filename[filename_size];

        //construct the correct path for the file and save it in the
        char array
        sprintf(buffer_of_filename,"%s/%s",argv[1],filename);

        //open the content of the array us directory
        not_file = opendir (buffer_of_filename);

        //if not open is a file and print the correct message

```

```

if( not_file != NULL)
{
    printf("\tThe %s is not a file\n",de->d_name);
    //return to the while to continue the reading of the folder
    continue;
}

//open for read only the file and check if opens
if ((fd=open(buffer_of_filename, O_RDONLY ))==-1)
{
    //Print to user the corresponding message
    perror("");
    //return to the while to continue the reading of the folder
    continue;
}

//find the size of the file by going to the pointer at the end
of the text to read all the characters
long filesize=lseek(fd, 0, SEEK_END);

//check if file is empty and print the correct message
if( filesize ==0 )
{
    printf("\tThe file is empty.\n");
    //return to the while to continue the reading of the folder
    continue;
}

//a char array for saving the character that read from the file
char buffer[filesize];

//set again the pointer of the file in the beginning to start
the reading
lseek(fd, 0, SEEK_SET);

//read the file and save the characters int the char array
characters=read(fd, buffer, filesize);

//check if is in ASCII
if(isnot_ascii(filesize,buffer))
{
    //return to the while to continue the reading of the folder

```

```

        continue;
    }

    //a counter to know how many children i have
    count_for_fork++;

    //create child process
    pid = fork();

    //check if created child process
    if (pid == -1)
    {
        //Print to user the corresponding message
        perror("");
        //exiting from the program
        exit(1);
    }

    //creates the correct arguments array to call exec
    char *args[]={"./wordcounter",buffer_of_filename,NULL};

    //for the parent
    if (pid != 0)
    {
        //Ignore SIGTERM and SIGINT signals
        signal(SIGINT, sig_Handler);
        signal(SIGTERM,sig_Handler);

    }else{

        //close the file
        close(fd);

        //go to the program that is the first element of the array
        execv(args[0],args);
    }
}

for(int i=0;i<count_for_fork;i++)
{
    //Wait for children
    wait(NULL);
}

```

```

    }
    //close correctly the folder
    closedir(pDir);

    //print a message to user to understand where is the output tha
    written by the program
    printf("\t.....Only files in Ascii is in the file output.txt.....\n");

    return 0;
}

wordcounter.c

#include "wordcounter.h"
#include <pthread.h>

//initialise the locks for threads
pthread_mutex_t mymutex=PTHREAD_MUTEX_INITIALIZER;

//initialise the total sum for the threads of one file
int sum=0;

//check if the given character is one from the separators of a word
int is_separator(char ch)
{
    if(ch=='\t' || ch=='\n' || ch=='\0' || ch==' ')
    {
        return 1;//true
    }
    return 0;//false
}

//counts the words of a file with the help of the threads
void *word_counter(void *structs_args)
{
    //a struct that contains the information for one thread
    info *args;
    args =structs_args;

    //a array two save two by two the characters that read by the read
function
    char characters[2];

    int fd;

```

```

if ((fd=open(args->filename, O_RDONLY ))==-1)
{
    //Print to user the corresponding message
    perror("");
    //return to the while to continue the reading of the folder
    exit(1);
}

//the number of the thread
int limit=args->counter_thread;

//counter for words that read a thread
int counter=0;
int i;

//calculate the size of the file to do right the sharing for threads
long filesize=lseek(fd,0,SEEK_END);

//set again the pointer of file in the beggining of the file
lseek(fd,0,SEEK_SET);

//give the correct offset of file descriptor for each thread
lseek(fd, limit * (filesize/NTHREADS), SEEK_SET );

//check if is the last thread to read correctly the lasts characters
in buffer
if(limit==NTHREADS-1 && filesize%NTHREADS!=0)
{
    //define the piece that will read the last thread
    for(int i=limit *
(filesize/NTHREADS);i<((limit+1)*(filesize/NTHREADS))+(filesize %
NTHREADS)-1;i++)
    {
        //read two by two characters to do the comparison and find
if something is word
        read(fd, characters, 2);

        //check if it is in the end of a word
        if(!is_separator(characters[0]) &&
is_separator(characters[1]))
        {

```



```

        //increase the counter of the words that read
        counter++;

    }

    /*
    Set the pointer of the file one character back to compare
all
    the right possible combinations to see if something is a
word.
    file-> abcd  what I want to compare each time-> ab bc
cd ....
    */
    lseek(fd, -1, SEEK_CUR );
}
else
{
    //define the piece that will read the last thread
    for(int i=limit *
(filesize/NTHREADS);i<((limit+1)*(filesize/NTHREADS));i++)
    {

        //read two by two characters to do the comparison and find
if something is word
        read(fd, characters, 2);

        //check if it is in the end of a word
        if(!is_separator(characters[0]) &&
is_separator(characters[1]))
        {
            //increase the counter of the words that read
            counter++;

        }

    }

    /*
    Set the pointer of the file one character back to compare
all
    the right possible combinations to see if something is a
word.
    file-> abcd  what I want to compare each time-> ab bc
cd ....
    */
    lseek(fd, -1, SEEK_CUR );
}
}
}

```

```

    }
}

//lock this area to protect it and write in sum only one thread each
time
pthread_mutex_lock(&mymutex);

//total sum of words that each thread read
sum=sum+counter;

//unlock this area
pthread_mutex_unlock(&mymutex);

//close correct the folder because when i recall this function i
open it again
close(fd);

//exit from the current thread
pthread_exit (NULL);
}

int main(int argc, char *argv[])
{
    //initialize the array of the threads
    pthread_t threads[NTHREADS];
    //initialize the struct of the arguments that every thread must has
    info counter[NTHREADS];
    int fd, characters;

    //for the threads
    for (int i = 0; i < NTHREADS; ++i)
    {
        //first info is the filename of the file that thread must open
        counter[i].filename=argv[1];

        //second info is the number of thread
        counter[i].counter_thread=i;

        //create the threads and call the correct function
        pthread_create(&threads[i], NULL, &word_counter, (void
        *)&(counter[i]));
    }
}

```

```

    }

    //wait the threads to complete
    for (int i = 0; i < NTHREADS; ++i)
    {
        pthread_join(threads[i],NULL);
    }

    //calculate the words of the file that will write in the file (6-
    >words:) + , , +\n=9
    int saveddata_size=(sizeof(getpid())+strlen(argv[1])+9+sizeof(sum));
    char buffer_of_saveddata[saveddata_size];

    //open the out.txt for write and append and create if does not exist
    and check if opens
    if ((fd=open("output.txt",O_WRONLY | O_CREAT | O_APPEND,0644 ))==-1)
    {
        //Print to user the corresponding message
        perror("");
        //exiting from the program
        exit(1);
    }

    //construct the correct sentence that will write in the file
    sprintf(buffer_of_saveddata,"%d, %s, Words:%d\n",getpid(),argv[1],sum);

    //write the array of characters in the file
    write(fd, buffer_of_saveddata, strlen(buffer_of_saveddata));

    return 0;
}

```

wordcounter.h

```

#ifndef WORDCOUNTER_H
#define WORDCOUNTER_H

#define NTHREADS 4

//common includes for both main
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/file.h>
#include <unistd.h>

```

```
typedef struct infoR info;
struct infoR
{
    char *filename;//the file that thread must read
    int counter_thread;//the number of the thread
};

//check if a character is a separator for words
int is_separator(char );

//calculate the words from a file
void *word_counter(void *);

//check if a character is not in ASCII
int isnot_ascii(int ,char *);

//to handle the sigint sigterm
void sig_Handler(int ) ;

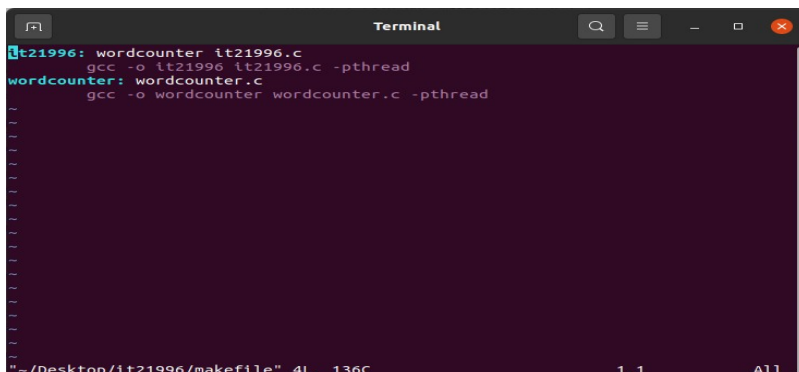
#endif
```

**Ακολουθούν τα
screenshots**

Τρόπος Εκτέλεσης

- ✓ Εκτέλεση του **make** :

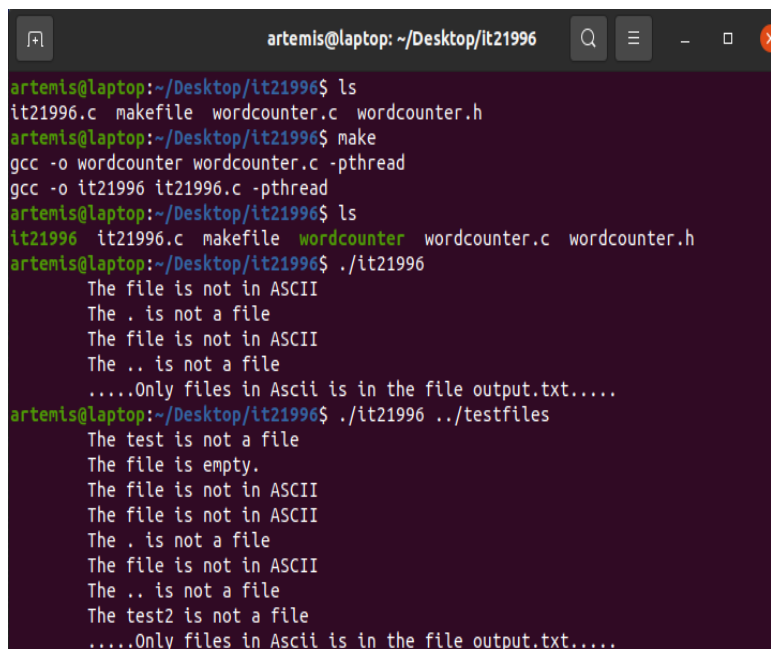
Για να μπορέσουμε να εκτελέσουμε την εντολή **make** και να παράξει το εκτελέσιμο αρχείο θα πρέπει να έχουμε στον φάκελο που είναι και η εργασία μας ένα **makefile** για να γίνουν **compile** και οι δύο **main** (μιας και χρησιμοποιώ την **exes** που χρειάζεται το εκτελέσιμο του **wordcounter.c** αρχείου για να μπορέσει να τρέξει σωστά).



```
it21996: wordcounter it21996.c
gcc -o it21996 it21996.c -pthread
wordcounter: wordcounter.c
gcc -o wordcounter wordcounter.c -pthread
```

Περιεχόμενα
makefile

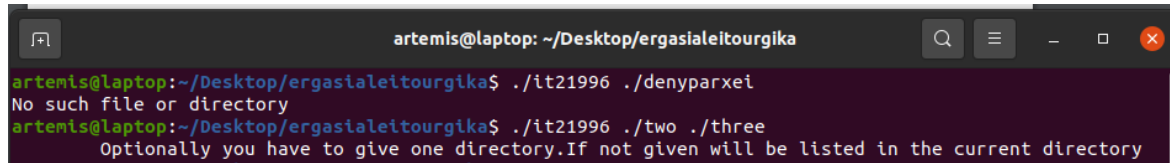
- ✓ Αφού υλοποιήσαμε αυτό το βήμα ,μετά μέσα στο φάκελο που είναι αυτό που θέλουμε να τρέξουμε καθώς και το **makefile** αρχείο εκτελούμε την εντολή **make** για να μας κάνει **compile** και τις δύο **main**.Στη συνέχεια , το εκτελέσιμο θα είναι το **it21996** και έτσι χρησιμοποιούμε την εντολή **./it21996 'ΌνομαΚαταλόγου** ή την **./it21996** αν θέλουμε να τρέξουμε το πρόγραμμα στον φάκελο που είμαστε.



```
artemis@laptop: ~/Desktop/it21996
artemis@laptop:~/Desktop/it21996$ ls
it21996.c makefile wordcounter.c wordcounter.h
artemis@laptop:~/Desktop/it21996$ make
gcc -o wordcounter wordcounter.c -pthread
gcc -o it21996 it21996.c -pthread
artemis@laptop:~/Desktop/it21996$ ls
it21996 it21996.c makefile wordcounter wordcounter.c wordcounter.h
artemis@laptop:~/Desktop/it21996$ ./it21996
The file is not in ASCII
The . is not a file
The file is not in ASCII
The .. is not a file
.....Only files in Ascii is in the file output.txt.....
artemis@laptop:~/Desktop/it21996$ ./it21996 ../testfiles
The test is not a file
The file is empty.
The file is not in ASCII
The file is not in ASCII
The . is not a file
The file is not in ASCII
The .. is not a file
The test2 is not a file
.....Only files in Ascii is in the file output.txt.....
```

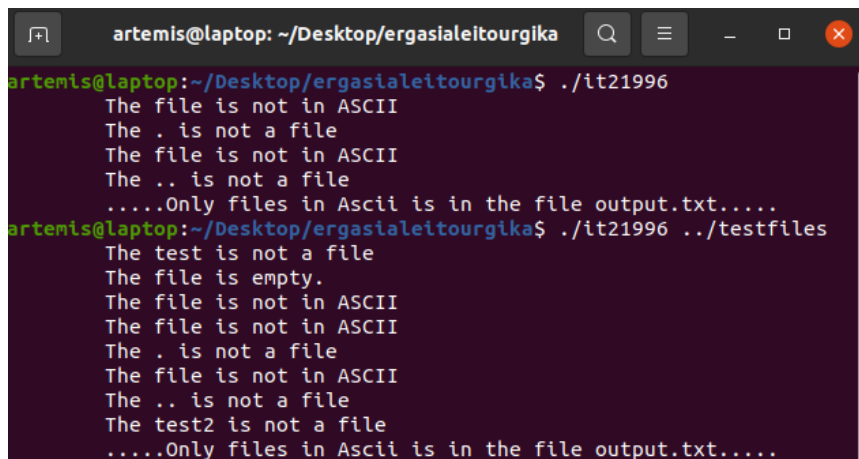
Ενδεικτικές εκτελέσεις (screenshots):

- ☐ Λάθος είσοδο από το χρήστη



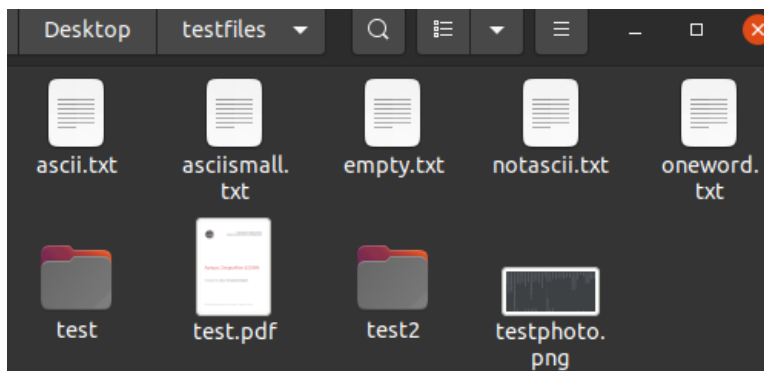
```
artemis@laptop: ~/Desktop/ergasialeitourgika
artemis@laptop:~/Desktop/ergasialeitourgika$ ./it21996 ./denyparxei
No such file or directory
artemis@laptop:~/Desktop/ergasialeitourgika$ ./it21996 ./two ./three
Optionally you have to give one directory.If not given will be listed in the current directory
```

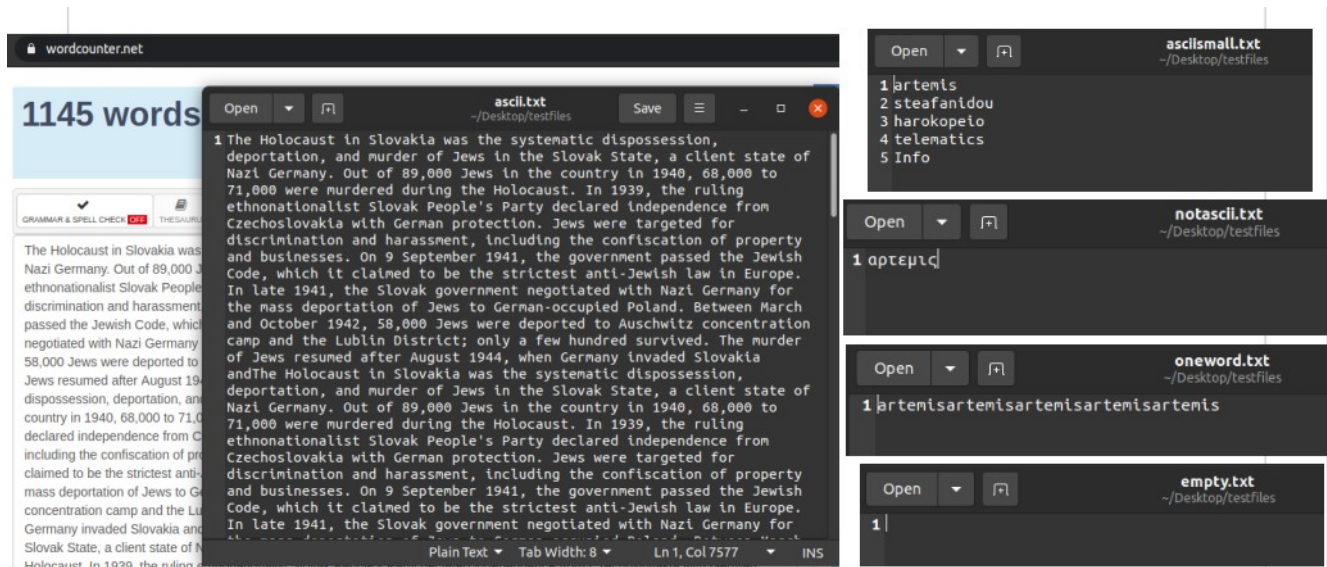
- ☐ Εκτέλεση με επιτυχία



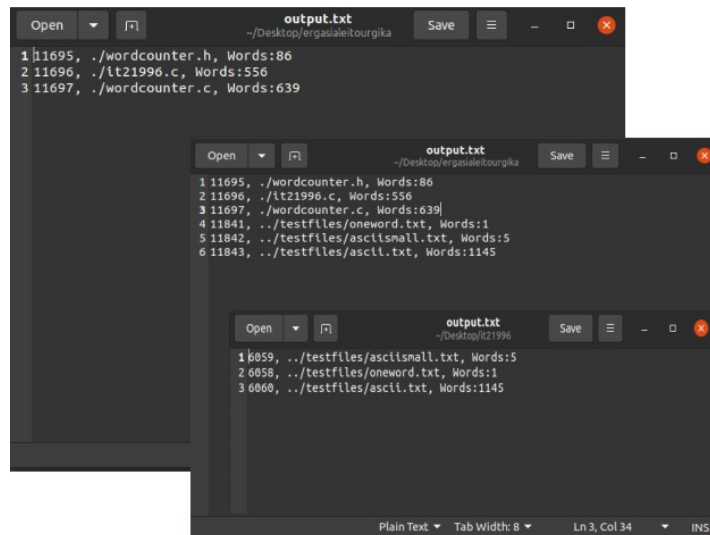
```
artemis@laptop:~/Desktop/ergasialeitourgika$ ./it21996
The file is not in ASCII
The . is not a file
The file is not in ASCII
The .. is not a file
.....Only files in Ascii is in the file output.txt.....
artemis@laptop:~/Desktop/ergasialeitourgika$ ./it21996 ../testfiles
The test is not a file
The file is empty.
The file is not in ASCII
The file is not in ASCII
The . is not a file
The file is not in ASCII
The .. is not a file
The test2 is not a file
.....Only files in Ascii is in the file output.txt.....
```

- ☐ Τα περιεχόμενα του **testfiles**





Πρώτη εκτέλεση του προγράμματος



Δεύτερη εκτέλεση του προγράμματος

Μία άλλη εκτέλεση του προγράμματος για να αποδείξουμε ότι δε γνωρίζουμε ποιο παιδί θα τελειώσει πρώτο αφού τα rid των παιδιών δεν βγαίνουν πάντα με τη σειρά

Παρατηρήσεις/σχόλια

Στην εκφώνηση αναφέρεται :

“Αποτελέσματα με τη μορφή:

<pid>, <filename measured>, <total_number_of_words> στο αρχείο output.txt”

Για να βλέπουμε απευθείας τον αριθμό των λέξεων και να μην μπερδευτούμε στο αρχείο output.txt πρόσθεσα και τη λέξη “words:” ελπίζοντας να μην υπάρχει ιδιαίτερο πρόβλημα.

Τελική μορφή output.txt: <pid>, <filename measured>, Words:<total_number_of_words>

- Δημιουργία νέων διεργασιών και φυσιολογικός τερματισμός τους

Παρατηρήσεις/σχόλια

```
int count_for_fork=0;

//read one per one the contents of the folder
while ((de = readdir(pDir)) != NULL)
{
    //a counter to know how many child i have
    count_for_fork++;

    //create child process
    pid = fork();

    //check if created child process
    if (pid == -1)
    {
        //Print to user the corresponding message
        perror("");
        //exiting from the program
        exit(1);
    }

    //creates the correct arguments array to call exec
    char *args[]={"./wordcounter",buffer_of_filename,NULL};

    //for the parent
    if (pid != 0)
    {
        //Ignore SIGTERM and SIGINT signals
        signal(SIGINT, sig_Handler);
        signal(SIGTERM, sig_Handler);

    }else{

        //close the file
        close(fd);

        //go to the program that is the first element of the array
        execv(args[0],args);
    }
}

for(int i=0;i<count_for_fork;i++)
{
    //Wait for children
    wait(NULL);
}
```

Με τη βοήθεια της **fork()** δημιουργούμε μία νέα διεργασία που είναι παιδί της βασικής μας διεργασίας(πατέρας).Το παιδί κληρονομεί τα περιεχόμενα του πατέρα παίρνοντας ένα αντίγραφο της μνήμης και έτσι οι δύο διεργασίες εκτελούνται από την επόμενη εντολή της **fork()**.Στην προκειμένη περίπτωση θέλουμε το παιδί να γνωρίζει τον file που θα πρέπει να διαβάσει με τη βοήθεια των **threads** και θέλουμε να χρησιμοποιήσουμε τόσες **fork()** όσες και τα διαφορετικά αρχεία που έχουμε διαβάσει από τον φάκελο γι αυτό και την **fork()** την έχουμε μέσα στο **while** που διαβάζουμε τον folder.

Για να είμαστε σίγουροι ότι δημιουργήσαμε μία νέα διεργασία ελέγχουμε στη συνέχεια το **pid** που θα γυρίσει η **system call fork()**.Αμα επιστρέψει **-1** σημαίνει ότι απέτυχε και έτσι τυπώνουμε στον χρήστη το αντίστοιχο μήνυμα λάθους.Η **fork()** άμα πετύχει επιστέφει στον πατέρα το **pid** του παιδιού που κατάφερε να δημιουργήσει και στο παιδί το **0**.Έτσι στη συνέχεια ,υλοποιήθηκε ο κώδικας για το μπαμπά για **pid!=0** και για το παιδί με **pid==0**.

Για να βεβαιωθούμε πως ο πατέρας θα περιμένει όλα του τα παιδιά να τελειώσουν και δεν θα δημιουργηθεί κάποια διεργασία zombie έβαλα έναν counter για να μετράει το πόσες φορές εκτελέστηκε η **fork()** μιας και κάθε φορά που εκτελείτε δημιουργείτε και ένα νέο παιδί.Μετά υλοποίησα ένα **for** από το **0** μέχρι και το **counter-1** δηλαδή μέχρι να μην έχω κάποιο παιδί να περιμένω και έκανα ένα **wait(null)** για να μην περιμένω κάποιο συγκεκριμένο παιδί να τελειώσει μιάς και δεν μπορώ να γνωρίζω ποιο θα είναι αυτό το **pid** που θα πρέπει να περιμένω.Έτσι,έχω εξασφαλίσει πως κανένα παιδί δε γίνεται zombie δηλαδή όλες οι διεργασίες τερματίζουν φυσιολογικά.

- Δημιουργία νημάτων και φυσιολογικός τερματισμός τους.

(screenshot->αποσπάσματα από τον κώδικα)

```
//counts the words of a file with the help of the threads
void *word_counter(void *structs_args)
{
    //lock this area to protect it and write in sum only one thread ea
    pthread_mutex_lock(&mymutex);

    //total sum of words that each thread read
    sum=sum+counter;

    //unlock this area
    pthread_mutex_unlock(&mymutex);

    //close correct the folder because when i recall this function i c
    close(fd);

    //exit from the current thread
    pthread_exit (NULL);
}

int main(int argc, char *argv[])
{
    //initialize the array of the threads
    pthread_t threads[NTHREADS];
    //initialize the struct of the arguments that every thread must has
    info counter[NTHREADS];
    int fd, characters;

    //for the threads
    for (int i = 0; i < NTHREADS; ++i)
    {
        //first info is the filename of the file that thread must open
        counter[i].filename=argv[1];

        //second info is the number of thread
        counter[i].counter_thread=1;

        //create the threads and call the correct function
        pthread_create(&threads[i], NULL, &word_counter, (void *)&(counter[i]));
    }

    //wait the threads to complete
    for (int i = 0; i < NTHREADS; ++i)
    {
        pthread_join(threads[i],NULL);
    }
}
```

Παρατηρήσεις/σχόλια

Για να μπορέσουμε να υλοποιήσουμε την πολυνηματική ανάγνωση ενός αρχείου χρειάζεται η δημιουργία των νημάτων-threads. Έτσι, χρησιμοποίησα την `pthread_create(&threads[i], NULL, &word_counter, (void *)&(counter[i]))`. Η `create` παίρνει σαν πρώτο όρισμα το `id` του thread σαν τρίτο την συνάρτηση που θα τρέξει μετά(στην προκειμένη περίπτωση την `word_counter` που υπολογίζει τις λέξεις σε ένα αρχείο) και σαν τέταρτο ένα struct με το όνομα του αρχείου το οποίο θα διαβάσει(ένα συγκεκριμένο κομμάτι του) το thread καθώς και το ποιο thread είναι, το οποίο θα μας βοηθήσει στη συνέχεια στο να ορίσουμε το κομμάτι του αρχείου που θα διαβάζει το κάθε thread. Αναμένουμε να τελειώσουν όλα τα threads χρησιμοποιώντας την `pthread_join(threads[i],NULL)` μέσα σε ένα for που φτάνει μέχρι το πλήθος των threads. Το δεύτερο όρισμα της `join` είναι το αποτέλεσμα που θα μας δώσουν ένα ένα τα threads. Εδώ όμως είναι `NUL` μιας και έχω ορίσει μία global μεταβλητή `sum` στην οποία αλλάζω τιμή μέσα στη συνάρτηση

που καλεί η `create` (με προσοχή γιατί σε αυτήν τη μεταβλητή έχουν πρόσβαση όλα τα threads οπότε για να την προστατέψω και για να την αλλάζει κάθε φορά ένα thread σωστά έβαλα το `lock`) άρα μπορεί να την δει και η `main` συνάρτηση.

Μέσα στον κώδικα του thread πρέπει να χρησιμοποιούμε την `pthread_exit` για να κάνουμε exit από το thread που είμαστε αλλά και για να επιστρέψουμε την τιμή που όμως εδώ αυτό το όρισμα είναι null για το λόγο που αναλύσαμε και παραπάνω, αφού η `sum` μεταβλητή είναι global. Έτσι, χρησιμοποιώντας αυτά βεβαιωνόμαστε ότι έχουμε έναν σωστό τρόπο δημιουργίας threads και τον φυσιολογικό τερματισμό τους.

Με δυσκόλεψε :

➤ Αριθμός threads :

Στο μάθημα είχαμε πει πως τα threads σχετίζονται με τους πυρήνες και έτσι σαν αρχή αυτή τη σκέψη έψαξα και είδα ότι τα threads που μπορεί να υποστηρίξει ένα μηχάνημα και να τρέχουν παράλληλα εξαρτώνται από το ίδιο το μηχάνημα. Έτσι, για να βρω στο δικό μου μηχάνημα πόσα είναι τα threads που μπορώ να χρησιμοποιήσω για να πετύχω την παραλληλία που ζητάει η άσκηση, έτρεξα στο terminal την εντολή **lscpu**. Είδα, ότι έχω δύο πυρήνες και 2 threads για κάθε ένα από αυτούς άρα $\text{πυρήνες} * \text{threads} = 2 * 2 = 4 \text{ threads}$ συνολικά που θα χρησιμοποιήσω στην εργασία.

➤ Τα κομμάτια για τα threads :

Το πρόγραμμα ξεκίνησα να το υλοποιώ πρώτα δίνοντας του ένα file και χρησιμοποιώντας μόνο την main με τα threads. Όταν του έδωσα ένα αρχείο δεν μου υπολόγιζε σωστά τις λέξεις και κάνοντας print τα όσα διάβαζα συνειδητοποίησα πως δεν μου διαβάζει όλους τους χαρακτήρες που υπάρχουν στο αρχείο. Μετά από ψάξιμο κατάλαβα πως γι' αυτό ευθύνονταν τα όρια που έδινα στην for. Όταν είχα αρχεία με ζυγό αριθμό χαρακτήρων τότε διάβαζε όλους τους χαρακτήρες αφού το `filesize/NTHREADS` ήταν (αν έχουμε 1000 χαρακτήρες) $1000/4$ (πιο πάνω η ανάλυση) σύνολο: **250** χαρακτήρες που διάβαζε το κάθε thread. Όταν όμως ήταν μονός αριθμός τότε (ας υποθέσουμε πως είχαμε 1005 χαρακτήρες): $1003/4 = 250,75$. Άρα, κάθε thread έπαιρνε 250 χαρακτήρες $250 * 4 = 1000$ άρα δεν διαβάζονταν 3 χαρακτήρες. Έτσι, σκέφτηκα πως αφού έχω μόνο 4 threads και άρα για να μην έχω κανέναν χαρακτήρα που δε διαβάζεται θα πρέπει το filesize να είναι πολλαπλάσιο του 4. Αν όμως δεν είναι, το μέγιστο πλήθος χαρακτήρων που μπορώ να "χάσω" είναι το 3. Αυτό συμβαίνει γιατί αν είχα 4 χαρακτήρες που έμεναν το 4 είναι ακέραιο πολλαπλάσιο του 4 άρα θα μοιράζονταν σωστά αν είχα 5 το πέντε μέσα του κρύβει το 4 άρα μοιράζονται σωστά αυτοί οι χαρακτήρες και μένει ένας, αν είχα 6 : $6 - 4 = 2$ που μένουν, αν είχα 7 : $7 - 4 = 3$ μένουν αν είχα 8 : $8 - 4 = 4 - 4 = 0$ άρα δεν μένει κάποιος χαρακτήρας. Στη συνέχεια, σκέφτηκα πως το 3 είναι πολύ μικρός αριθμός άρα θα μπορούσα να το προσθέσω στο τελευταίο thread και να είχα **250 250 250 253 αν 1003** σύνολο οι χαρακτήρες του αρχείου. Αν είχα παραπάνω threads θα μπορούσα να κάνω μία for από το 1 μέχρι το `filesize%NTHREADS` και να προσθέτω έναν παραπάνω χαρακτήρα σε κάθε thread μέχρι να τελειώσει το for.

➤ Έλεγχος άμα είναι αρχείο :

Η αλήθεια είναι πως αυτό με δυσκόλεψε περισσότερο από όλα μιας και δεν έβρισκα κάτι που να μπορώ να το καταλάβω. Έτσι, σκέφτηκα να χρησιμοποιήσω την `opendir` και άμα καταφέρει και ανοίξει κάτι πάει να πει πως αυτό είναι φάκελος, άρα δεν είναι αρχείο. Άρα, άμα δεν καταφέρει να το ανοίξει τότε λογικά θα είναι αρχείο.

Συνοπτικός Πίνακας

2η Εργασία		
Λειτουργία	Υλοποιήθηκε (ΝΑΙ/ΟΧΙ/ ΜΕΡΙΚΩΣ)	Συνοπτικές Παρατηρήσεις
Βασική διεργασία η οποία ελέγχει τις παραμέτρους και βρίσκει τα ascii αρχεία	ΝΑΙ	Τίποτα
Δημιουργία νέων διεργασιών και φυσιολογικός τερματισμός τους	ΝΑΙ	Τίποτα
Δημιουργία νημάτων και φυσιολογικός τερματισμός τους	ΝΑΙ	Τίποτα
Ομαλή εκτέλεση προγράμματος, error handling, τεκμηρίωση	ΝΑΙ	Τίποτα