



ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΜΑΤΙΚΗΣ

Τσεσμελής Μελέτιος

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

Ταύρος, 31 Ιανουαρίου 2021

| | |
|---|-----------|
| Περιεχόμενα | |
| Άσκηση 2 | 3 |
| Κώδικας | 3 |
| Τρόπος Εκτέλεσης | 13 |
| Ενδεικτικές εκτελέσεις (screenshots): | 14 |
| Βασική εκτέλεση του προγράμματος | 14 |
| Παρατηρήσεις/σχόλια | 15 |
| Βασική διεργασία η οποία ελέγχει τις παραμέτρους και βρίσκει τα ascii | 16 |
| Παρατηρήσεις/σχόλια | 17 |
| Δημιουργία νέων διεργασιών και φυσιολογικός τερματισμός τους | 18 |
| Παρατηρήσεις/σχόλια | 18 |
| Δημιουργία νημάτων και φυσιολογικός τερματισμός τους | 19 |
| Παρατηρήσεις/σχόλια | 19 |
| Γενικά Σχόλια/Παρατηρήσεις/Δυσκολίες | 24 |
| Συνοπτικός Πίνακας | 26 |

Άσκηση 2

Κώδικας

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

```
#include <stdio.h>                //it219105.c
#include <dirent.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>
#include <errno.h>
#include <signal.h>
#include <unistd.h>
#include <sys/stat.h>

//declare function, their implementation is on end
void handle_sigint(int);
void handle_sigterm(int);

int main (int argc, char *argv[])
{
    //check if he gave more than one argument.
    if (argc>2){
        printf("\tYou cannot give more than one arguments!");
        return -1;
    }
    DIR *folder;
    char *dir;
    //if he gave me directory save his path otherwise save current directory
    if (argc==2) //so he gave me an argument
    {
        //open given directory
        dir=argv[1];
    }else{
```

```

        //open current directory
        dir=".";
    }
    folder = opendir(dir);

    //check if directory opened successfully
    if(folder == NULL)
    {
        puts("\tUnable to read directory");
        exit(-1);
    }

    //declare variables before while, so i don't make unnecessary statements
    struct dirent *pde;           // Pointer for Directory Entry
    char * name_of_file;          //use it to take file of each name
    int name_of_file_has_size;    //use it to take size of each file name
    DIR* check_is_file;          //use it to check if it's file
    int fd ;                     //use it to open files
    int number_of_characters;    //use it to take the number of characters
    long file_size;              //use it to take the size of each file
    pid_t pid,ppid;              //use them to take pid and parent pid (ppid)
    int forks=0;                 //sum of forks, so i can check for zombies

    //I use readdir to read this directory
    while ((pde = readdir(folder)) != NULL)
    {
        printf("%s\n", pde->d_name);
        //so now i have the names of whatever is inside of folder, one under
the other.
        name_of_file= pde->d_name;
        //use strlen to find the size of string-name of file
        name_of_file_has_size=strlen(dir) +1/*for backslash*/
+strlen(name_of_file);
        //make a buffer for this string
        char buffer_for_name_of_file[name_of_file_has_size];
        //use sprintf to printf a string (for that is the s on start)
        sprintf(buffer_for_name_of_file,"%s/%s",dir,name_of_file);
        //use opendir to check if it is a file

```

```

check_is_file= opendir(buffer_for_name_of_file);
if(check_is_file!=NULL) // so it`s not a file, go on next
{
    printf("\t%s: it`s not a file!\n",pde->d_name);
    continue;
}

//open each file
fd=open(buffer_for_name_of_file, O_RDONLY); //we need Read only
if(fd<0) //has to be positive to be right
{
    perror("open");
    exit(1);
}
// printf("\tsuccessfully opened the fd=%d \n",fd);

//calculate file size with lseek use
file_size= lseek(fd,0,SEEK_END); //now cursor accessed the entire file
and stayed at end of it
if(file_size==0)
{
    printf("\tEmpty file!");
    continue; //go to check for next file, you have not words to count
here.
}
lseek(fd,0,SEEK_SET); //put cursor again on start of file

//create a buffer to help us read the number of characters
char buff[file_size];
//take the number of characters using read()
number_of_characters=read(fd,buff,file_size); //so now buffer has the
characters

```

```

/*
use buffer (buff) to take characters and
check if each of them is on the ASCII table.
If not then I print a message and I go on to the next file.
*/
int found_no_Ascii=0;

for(int i=0;i<file_size;i++)
{
    //printf("buffer for %d prints ->%d\n\n",i,buff[i]);
    if(buff[i]<0||buff[i]>127)//so it's out of ASCII
    {
        found_no_Ascii=1;
        i+=file_size-i; //so if you find one go on end, it's enough
    }
}
// printf("file size is %ld",file_size);
if(found_no_Ascii==1)
{
    printf("\tThis file it's not with ASCII characters!\n");
    continue; //go to check for other files
}

//end of first part of the exercise (basic process)

//start of creation of new processes

/*
I am going to use an exec function
before that I had to create an array with arguments
-> the path of counting-program
-> the buffer of file name
-> the buffer of that file
->null, to mark the end of the array of pointers
*/
char *args[]={"./counting_words",buffer_for_name_of_file,buff,NULL};

```

```

/*
I use fork to create a new process
for each file (if a file is on this line
then sure is on ascii)
*/
forks++;
pid=fork();

if(pid<0) //check for fail creation
{
    perror("creation of a child process was unsuccessful\n");
    exit(1);
}
//check if returned child or parent
if(pid==0)//child
{
    //use execv to replace the process with another, and count the
words of the file
    execv(args[0],args);
}else{ //parent
    //use signal for handling
    signal(SIGINT, handle_sigint);
    signal(SIGTERM,handle_sigterm);
}

//close file
if(close(fd)<0)
{
    perror("\tUnexpected error!Something went wrong with close.");
    exit(1);
}
}
//take care of zombies processes, wait to end all the children!
for(int i=0;i<forks;i++){
    wait(NULL);
}
closedir(folder);
exit(1);
return 0;
}

```

```
//FUNCTIONS FOR ERROR HANDLING
void handle_sigterm(int signal)
{
    printf("\n\tIgnored signal %d\n", signal);
}

void handle_sigint(int signal)
{
    printf("\n\tIgnored signal %d\n", signal);
}
```

```
                //Necessary code to run the program-compilation
                //counting_words.c

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <pthread.h>
#include <stdio.h>
#include <sys/file.h>
#include <unistd.h>
#include <string.h>
#include <error.h>

#define NTHREADS 8 /*maybe you have to change that, based on my system.
                    In case your system can't manage at the same time 8 threads
                    Then low that number. Or put a greater if your system and
                    cores can use them.
                    */

//make a struct to take number of thread and the buffer(so what it has to read)
typedef struct dataS data;
struct dataS
{
    char *buffer;//we use the buffer to take the characters
    int thread_counter;//to save the sum of threads
};
```



```

//initialize mutex
pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;

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


// DECLARE FUNCTION - IMPLEMENTATION IS ON END


    //check if the given character is one from the separators of a word
int word_seperator(char);
    //counts the words of a file with the help of the threads
void *counter_words_of_file(void *);


//MAIN FUNCTION
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
    data count[NTHREADS];

    //create threads
    for (int i = 0; i < NTHREADS; i++)
    {
        //first data is the characters from the file
        count[i].buffer=argv[2];
        //second data is the number of thread
        count[i].thread_counter=i;
        pthread_create(&threads[i], NULL, &counter_words_of_file, (void
*)&(count[i]));
    }
}

```

```

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

//calculate the size of output file
int size_of_output_file=(sizeof(getpid()+strlen(argv[1]) +5 /*for commas
and newline*/+sizeof(total_number));
char buffer_output[size_of_output_file];

int fd;
/*
    permissions is like
    r-- ->4          so i want 644 to give rw--w--w-
    -w- ->2
    --x -<1
*/
//create an output file and write on it... if already exists append that
fd=open("output.txt",O_WRONLY | O_CREAT | O_APPEND,0644 );
if (fd==-1)
{
    perror("open");
    exit(-1);
}

//prepare the format of text we will print on output file
sprintf(buffer_output,"%d, %s, %d\n",getpid(),argv[1],total_number);

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

return 0;
}

//FUNCTIONS IMPLEMENTATIONS
/*I take as words only patterns of letters. For example: "meletis !" is one
Word or "meletis 219105!" is one word. The reason is that I want more realistic
and meaningful counting and I don't want take as words syntax mistakes of user
Like "Hello my friend , today ..." commas etc isn't words.*/

```

```

int word_seperator(char character)
{
    if(character==' ' || character=='\n' || character=='\t' || character=='\0'
//this line (first) is based on online program "word counter"
    || character==',' || character=='!' || character=='`' || character=='('
//if you want to check it with that a specific text
    || character==')' || character=='{' || character=='}' || character=='['
//then ignore other lines
    || character==']' || character=='+' || character=='-' || character=='>'
    || character=='<' || character=='.' || character=='%' || character=='@'
    || character=='*' || character=='^' || character=='*' || character=='"'
    || character=='~' || character=='/' || character==';' || character=='&'
    || character=='#' || character=='$' || character=='_' || character==' ':'
    || character=='0' || character=='1' || character=='2' || character=='3'
    || character=='4' || character=='5' || character=='6' || character=='7'
    || character=='?' || character=='8' || character=='9' || character=='\'\'
    || character=='|')
    {
        return 1;//true,is a seperator
    }
    return 0;//false,its not a seperator
}

//counts the words of a file with the help of the threads
void *counter_words_of_file(void *data_args)
{
    //take the data of the thread
    data *args;
    args =data_args;

    //take the number of threads
    int number_of_threads=args->thread_counter;

    //make a counter for words of a thread
    int counter=0;
    // int i;

    //check if is the last thread to read correctly the lasts characters in
    buffer
    int modulo=0;

```

```

        if(number_of_threads==NTHREADS-1 && strlen(args->buffer)%NTHREADS!=0){
            modulo=(strlen(args->buffer)%NTHREADS)-1;
        }
        for(int
i=number_of_threads*(strlen(args->buffer)/NTHREADS);i<(number_of_threads+1)*(st
rlen(args->buffer)/NTHREADS)+modulo;i++)
        {
            //check if it is in the end of a word
            if(!word_seperator(args->buffer[i]) &&
word_seperator(args->buffer[i+1]))
            {
                //add by one the counter of words
                counter++;
            }
        }

        // "freeze" threads, to control threads we have to let one thread at the
time
        pthread_mutex_lock(&mutex);

        //and then let that thread to add on sum of counter
        total_number+=counter;

        //and unlock - unfreeze threads
        pthread_mutex_unlock(&mutex);

        //exit of thread
        pthread_exit (NULL);
    }

```

Τρόπος Εκτέλεσης

Ξεκινώντας με τον τρόπο εκτέλεσης, λόγω του ότι χρησιμοποιείται μια συγγενική εντολή της `exec` (η `execv`) όπως παρατηρούμε και παραπάνω χρησιμοποιούμε ένα βοηθητικό πρόγραμμα (`counting_words.c`). Για να επιτευχθεί το compilation του “βασικού” κώδικα, δημιουργήθηκε ένα `makefile` μέσω από το οποίο τρέχει το βοηθητικό πρώτα και μετά το βασικό πρόγραμμα.

Σχετικά με το `makefile`, έπεται από μερικά tutorials για το τι πρέπει να περιέχει κατέληξα στα εξής:

| makefile | output.txt |
|---|------------|
| <pre>it219105: counting_words it219105.c gcc it219105.c -o it219105 -pthread counting_words: counting_words.c gcc counting_words.c -o counting_words -pthread</pre> | |

Παράδειγμα εκτέλεσης: (`make` → `./it219105` →(ή) `./it219105` (directory))

```
meletis@meletis-VirtualBox:~/Desktop/word_counder_in_c$ make
gcc counting_words.c -o counting_words -pthread
gcc it219105.c -o it219105 -pthread
meletis@meletis-VirtualBox:~/Desktop/word_counder_in_c$ ./it219105
it219105
This file it's not with ASCII characters!
counting words.c
This file it's not with ASCII characters!
README.md
it219105.c
.
.: it's not a file!
makefile
counting_words
This file it's not with ASCII characters!
..
.: it's not a file!
meletis@meletis-VirtualBox:~/Desktop/word_counder_in_c$ ./it219105 ../files
fileTwo.txt
.
.: it's not a file!
fileThree.txt
This file it's not with ASCII characters!
fileFour.txt
fileTwo (another copy).txt
fileTwo (copy).txt
..
.: it's not a file!
meletis@meletis-VirtualBox:~/Desktop/word_counder_in_c$
```

Ενδεικτικές εκτελέσεις (screenshots):

☐ Βασική εκτέλεση του προγράμματος

- ☐ Με όρισμα ένα directory

```
meletis@meletis-VirtualBox:~/Desktop/it219105$ ./it219105 ../files
fileTwo.txt
.
    ..: it`s not a file!
fileThree.txt
    This file it`s not with ASCII characters!
fileFour.txt
file.txt
..
    ..: it`s not a file!
```

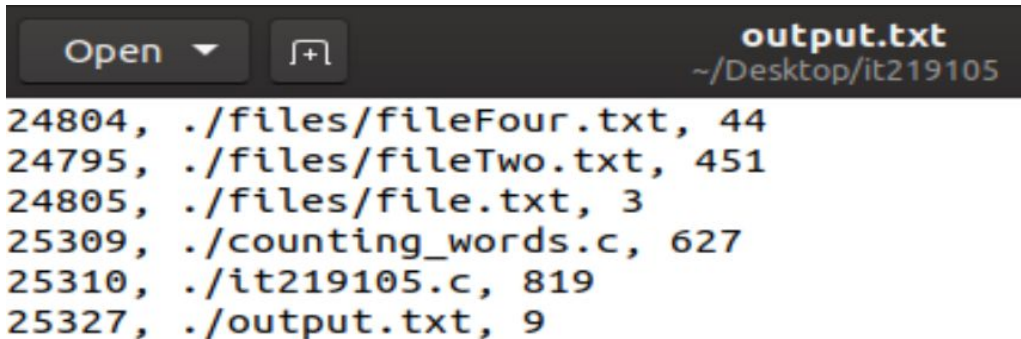
Έξοδος στο output.txt

```
Open ▾ [icon] output.txt
~/Desktop/it219105
24804, ./files/fileFour.txt, 44
24795, ./files/fileTwo.txt, 451
24805, ./files/file.txt, 3
```

- ☐ Χωρίς όρισμα

```
meletis@meletis-VirtualBox:~/Desktop/it219105$ ./it219105
it219105
    This file it`s not with ASCII characters!
counting_words.c
it219105.c
.
    ..: it`s not a file!
output.txt
counting_words
    This file it`s not with ASCII characters!
files
    files: it`s not a file!
..
    ..: it`s not a file!
meletis@meletis-VirtualBox:~/Desktop/it219105$
```

Έξοδος στο output.txt



```
24804, ./files/fileFour.txt, 44
24795, ./files/fileTwo.txt, 451
24805, ./files/file.txt, 3
25309, ./counting_words.c, 627
25310, ./it219105.c, 819
25327, ./output.txt, 9
```

Παρατηρήσεις/σχόλια

Το directory ./files που βλέπουμε στο screenshot περιέχει 4 αρχεία (txt) τα οποία δημιούργησα με σκοπό να ξέρω τον αριθμό των λέξεων. Συγκεκριμένα είναι τα : file.txt, fileTwo.txt, fileThree.txt, fileFour.txt στο output.txt φαίνονται τα 3 από αυτά καθώς όπως παρατηρούμε στο terminal (χρησιμοποιείται το VS) το fileThree.txt περιέχει έστω και έναν χαρακτήρα που δεν είναι στο ASCII επομένως δεν προβαίνουμε στην καταμέτρηση των λέξεων.

Αξίζει να σημειωθεί πως κατά την εκτέλεση στο terminal αριστερά εκτυπώνεται το όνομα του αρχείου που ελέγχουμε και από κάτω και με ένα tab (\t) εκτυπώνεται αν υπάρχει κάποιο μήνυμα λάθους ή απόρριψης αντίστοιχα. Επίσης έχουν δημιουργηθεί αυτόματα 2 κρυφά αρχεία με το όνομα "." και ".." τα οποία απορρίπτονται όπως φαίνεται.

Σχετικά με τα αποτελέσματα που βγαίνουν στο output.txt όπως είπαμε είναι από αρχεία που δημιούργησα επομένως στα πρώτα 3 που φαίνονται (έξοδος, χωρίς όρισμα) η διαδικασία και τα αποτελέσματα βγαίνουν σωστά. Θα μπορούσαμε να σχολιάσουμε το τελευταίο που είναι το ίδιο το output.txt. Άρα καταμετρά από εκείνο που φαίνεται στο screenshot όταν βάλαμε το όρισμα.

Σε αυτό το σημείο πρέπει να σχολιάσω ότι για να αποφευχθεί η καταμέτρηση σημείων στίξεων ή αριθμών ως λέξεις π.χ " it2191 05 !" αυτό εκλαμβάνεται ως μία λέξη ή το "Μελέτης !@#\$ Τσεσμελής!" ως δύο λέξεις. Με λίγα λόγια οι μη λατινικοί χαρακτήρες λαμβάνονται ως separators. Είναι επιλογή προτίμησης μιας και δεν περιορίστηκε και απλά ζητήθηκε να αγνοούμε τα κενά τα οποία φυσικά παραλείπονται. Για αυτό προκύπτει 9 λέξεις το output.txt όπως φαίνεται. Επομένως είναι ακόμα ένα σημείο επαλήθευσης της ορθής λειτουργίας.

Συνάρτηση για τα separators που υπάρχει στον κώδικα για διευκόλυνση σε τυχόν ελέγχους:


```

int word_seperator(char character)
{
    if(character==' ' || character=='\n' || character=='\t' || character=='\0'
    || character==',' || character=='!' || character=='.' || character=='('
    || character=='\'' || character=='{' || character=='}' || character=='['
    || character==']' || character=='+' || character=='-' || character=='>'
    || character=='<' || character=='.' || character=='%' || character=='@'
    || character=='*' || character=='^' || character=='"' || character=='"'
    || character=='~' || character=='/' || character==';' || character=='&'
    || character=='#' || character=='$' || character==' ' || character=='.'
    || character=='0' || character=='1' || character=='2' || character=='3'
    || character=='4' || character=='5' || character=='6' || character=='7'
    || character=='?' || character=='8' || character=='9' || character=='\')
    {
        return 1; //true, is a separator
    }
    return 0; //false, it's not a separator
}

```

❑ Βασική διεργασία η οποία ελέγχει τις παραμέτρους και βρίσκει τα ascii αρχεία

Σε αυτό το μέρος κληθήκαμε να υλοποιήσουμε ελέγχους υπεύθυνους για τις παραμέτρους και για το διάβασμα ASCII αρχείων.

Το τμήμα κώδικα για τους ελέγχους εισόδου και το ASCII που εκτελέστηκε είναι:

```

//check if he gave more than one arguments.
if (argc>2){
    printf("\tYou cannot give more than one arguments!");
    return -1;
}
DIR *folder;
char *dir;
//if he gave me directory save his path otherwise save current directory
if (argc==2) //so he gave me an argument
{
    //open given directory
    dir=argv[1];
}else{
    //open current directory
    dir=".";
}
folder = opendir(dir);
//check if directory opened successfully
if(folder == NULL)
{
    puts("\tUnable to read directory");
    exit(-1);
}
while ((pde = readdir(folder)) != NULL)
{
    printf("%s\n", pde->d_name);
    //so now i have the names of whatever is inside of folder, one under the other.
    name_of_file= pde->d_name;
    //use strlen to find the size of string-name of file
    name_of_file_has_size=strlen(dir) +1/*for new line*/ +strlen(name_of_file);
    //make a buffer for this string
    char buffer_for_name_of_file[name_of_file_has_size];
    //use sprintf to printf a string (for that is the s on start)
    sprintf(buffer_for_name_of_file,"%s/%s",dir,name_of_file);
    //use opendir to check if it is a file
    check_is_file= opendir(buffer_for_name_of_file);
    if(check_is_file==NULL) // so it's not a file, go on next
    {
        printf("\t%s: it's not a file!\n",pde->d_name);
        continue;
    }

    //open each file
    fd=open(buffer_for_name_of_file, O_RDONLY); //we need Read only
    if(fd<0) //has to be possitive to be right
    {
        perror("open");
        exit(1);
    }
    // printf("\tsuccessfully opened the fd=%d \n",fd);

    //calculate file size with lseek use
    file_size= lseek(fd,0,SEEK_END); //now cursor accessed the entire file and stayed at end of it
    if(file_size==0)
    {
        printf("\tEmpty file!");
        continue; //go to check for next file, you have not words to count here.
    }
    lseek(fd,0,SEEK_SET); //put cursor again on start of file
    //create a buffer to help us read the number of characters
    char buff[file_size];
    //take the number of characters using read()
    number_of_characters=read(fd,buff,file_size); //so now buffer has the characters

    /*
    use buffer (buff) to take characters and
    check if each of them is on ASCII table.
    if not then i print a message and i go on next file.
    */
    int found_no_Ascii=0;

    for(int i=0;i<file_size;i++)

```


Παρατηρήσεις/σχόλια

Συγκεκριμένα, γίνεται έλεγχος σχετικά με το αν δόθηκαν το πολύ 2 όρισμα (1 για να εκτελεστεί και 1 για το directory). Σε περίπτωση που δοθούν παραπάνω εμφανίζεται ένα κατάλληλο μήνυμα λάθους και γίνεται έξοδος από το πρόγραμμα.

Αν δοθεί λοιπόν κάποιο argument (εκτός εκείνου για την εκτέλεση) τότε με το σκεπτικό το ότι αν είναι directory θα το ανοίξουμε για να επιτευχθεί η διαδικασία καταμέτρησης λέξεων σε κάθε αρχείο του, θεωρήθηκε καλύτερο να ελεγχθεί αν μπορεί να ανοιχθεί απευθείας ως directory.

Επομένως χρησιμοποιείται η opendir() η οποία επιστρέφει null αν δεν τα καταφέρει, επομένως τότε είτε δεν θα έχουμε δικαίωμα να το διαβάσουμε είτε δεν θα είναι directory οπότε σε κάθε περίπτωση διακόπτεται το πρόγραμμα.

Αν ανοίξει τότε όσο υπάρχει κάτι να διαβάσει μέσα στον κατάλογο (readdir()), τότε διαβάζεται κάθε γραμμή και ελέγχεται αν είναι αρχείο. Για αυτό χρησιμοποιείται ένας buffer ο οποίος παίρνει σαν String (pattern χαρακτήρων) και ελέγχεται με την ίδια τεχνική για το αν είναι file, αν μπορέσουμε και το ανοίξουμε. Διαφορετικά γίνεται continue δηλαδή πάμε στον έλεγχο της επόμενης σειράς-String άρα και εν δυνάμει φακέλου προς έλεγχο.

Έπειτα χρησιμοποιείται η open() για να ανοιχτεί το αρχείο σε read only μορφή, χρησιμοποιείται και η perror() σε περίπτωση που κάτι πάει λάθος.

Εφόσον ανοιχτεί, γίνεται έλεγχος χρησιμοποιώντας την lseek το μέγεθος του. Αν είναι άδαιο τότε δεν χρειάζεται να ξεκινήσουμε διαδικασίες καταμέτρησης εφόσον δεν έχουμε κάτι να καταμετρήσουμε άρα συνεχίζουμε στον επόμενο έλεγχο, αν υπάρχει. Να σημειωθεί ότι επειδή η lseek μετακίνησε τον κέρσορα στο τέλος του αρχείου χρειάστηκε να το επαναφέρουμε στην αρχή προκειμένου να δημιουργηθεί ένας buffer ακόμα και με την read() να διαβάσει τους χαρακτήρες του αρχείου.

Τέλος χρησιμοποιείται μια μεταβλητή σαν Boolean προκειμένου όσο περνάμε και διαβάζουμε γράμμα γράμμα το αρχείο να ελέγχονται αυτά και αν βρεθεί έστω ένας εκτός ascii τότε μεταφέρουμε την τιμή του i σε τιμή τέτοια που να βγει απευθείας από το loop γλυτώνοντας άσκοπους ελέγχους και περνάμε στην μεταβλητή ότι βρέθηκε ένας “μη-ascii” χαρακτήρας.

❏ Δημιουργία νέων διεργασιών και φυσιολογικός τερματισμός τους

```
char *args[]={ ./counting_words ,buffer_for_name_of_file,buffer,NULL};

/*
I use fork to create a new process
for each file (if a file is on this line
then sure is on ascii)
*/
forks++;
pid=fork();

if(pid<0) //check for fail creation
{
    perror("creation of a child process was unsuccessful\n");
    exit(1);
}
//check if returned child or parent
if(pid==0)//child
{
    //use execv to replace the processes with another, and count the words of the file
    execv(args[0],args);
}else{//parent
    //use signal for handling
    signal(SIGINT, handle_sigint);
    signal(SIGTERM,handle_sigterm);
}

//close file
if(close(fd)<0)
{
    perror("\tUnexpected error!Something went wrong with close.");
    exit(1);
}

//take care for zombies processes, wait to end all the children!
for(int i=0;i<forks;i++){
    wait(NULL);
}
closedir(folder);
exit(1);
return 0;

//FUNCTIONS FOR ERROR HANDLING
void handle_sigterm(int signal)
{
    printf("\n\tIgnored signal %d\n", signal);
}

void handle_sigint(int signal)
{
    printf("\n\tIgnored signal %d\n", signal);
}
```

Παρατηρήσεις/σχόλια

Σε αυτό το μέρος της διεργασίας κληθήκαμε να δημιουργήσουμε νέες διεργασίες και όταν εκτελέσουν το έργο τους να καταφέρουμε να τις κλείσουν ομαλά χωρίς να υπάρξουν διεργασίες Zombie.

Συγκεκριμένα, χρησιμοποιήθηκε μια fork για κάθε αρχείο. Ελέγχεται αν η κλήση της fork ήταν ανεπιτυχής (<0) και αν όχι, τότε αν είναι παιδί(==0) η πατέρας(!=0). Αν είναι παιδί τότε χρησιμοποιείται η execv (vector -πίνακας) προκειμένου να αντικατασταθεί ο κώδικας (θα σχολιασθεί ο κώδικας με τα νήματα παρακάτω).

Σε περίπτωση που είναι πατέρας τότε χρησιμοποιείται η signal() για να αγνοήσουμε, να κάνουμε error handling των σημάτων SIGINT,SIGTERM με κατάλληλο μήνυμα που δίνουμε χρησιμοποιώντας δύο όμοιες συναρτήσεις για να εκτυπωθούν.

Στην συνέχεια κλείνουμε και ελέγχουμε για τον ορθό κλείσιμο του file distributor. Για να επιτευχθεί το να μη δημιουργούνται zombie χρησιμοποιείται μια for η οποία τρέχει για όσο υπάρχουν threads και εκτελεί μια wait() η οποία έχει οριστεί με NULL προκειμένου να περιμένει απλά να τελειώσουν όλα τα παιδιά.

❑ Δημιουργία νημάτων και φυσιολογικός τερματισμός τους

Σε αυτό το μέρος της εργασίας κληθήκαμε να υλοποιήσουμε νήματα τα οποία θα συγχρονίζονται (θα δουλεύουν ταυτόχρονα) και θα χωρίζουν τις εργασίες που πρέπει να γίνουν σε κάθε αρχείο για τον υπολογισμό των λέξεων.

Εφόσον όπως αναλύσαμε και παραπάνω έχουμε δημιουργήσει με την fork διεργασίες ανάλογα με τα πόσα files ascii υπάρχουν στο directory που εργαζόμαστε, στην περίπτωση όπως λέγαμε και παραπάνω που επιστραφεί παιδί χρησιμοποιήθηκε η execv (προκειμένου να περάσει τις πληροφορίες σε έναν πίνακα στο βοηθητικό πρόγραμμα). Στην πραγματικότητα η execv αντικαθιστά τη συνέχεια του προγράμματος με του βοηθητικού.

```
#define NTHREADS 8 /*maybe you have to change that, based in my system.
                    In case your system can't manage at the same time 8 threads
                    Then low that number. Or put a greater if your system can
                    use them
                    */

//make a struct to take number of thread and the buffer(so what it has to read)
typedef struct dataS data;
struct dataS
{
    char *buffer;//we use the buffer to take the characters
    int thread_counter;//to save the sum of threads
};

//initialize mutex
pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;

//initialize the total sum for the threads of one file
int total_number=0;
```

παρατηρήσεις/σχόλια

Αυτό που πρέπει να σχολιασθεί παραπάνω είναι ότι είναι σημαντικό τα threads που χρησιμοποιούνται να είναι ανάλογα με τις δυνατότητες του κάθε λειτουργικού συστήματος. Εγώ χρησιμοποίησα ένα με 8 cores επομένως είχα την δυνατότητα να επεξεργαστώ και 8 threads παράλληλα. Αν χρησιμοποιηθούν παραπάνω threads από ότι μπορεί το λειτουργικό τότε χάνεται η έννοια της παραλληλίας καθώς θα αναγκαστεί κάποιο να περιμένει για να ξεκινήσει δεν θα μπορέσει να ξεκινήσει με τα υπόλοιπα.

Στην συνέχεια δημιουργείται ένα struct για να μπορούμε να πάρουμε τους χαρακτήρες και τον αριθμό των threads Που θα χρειαστούμε στην πορεία. Γίνεται initialize του mutex (θα δούμε παρακάτω που θα το χρειαστούμε) και του αθροιστή που θα χρησιμοποιούν τα threads για να μετρήσουν τις λέξεις.Με άλλα λόγια το total_number πρέπει να έχει τον αριθμό των λέξεων.

```

//check if the given character is one from the separators of a word
int word_seperator(char);
//counts the words of a file with the help of the threads
void *counter_words_of_file(void *);

//MAIN FUNCTION
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
    data count[NTHREADS];

    //create threads
    for (int i = 0; i < NTHREADS; i++)
    {
        //first data is the characters from the file
        count[i].buffer=argv[2];
        //second data is the number of thread
        count[i].thread_counter=i;
        pthread_create(&threads[i], NULL, &counter_words_of_file, (void *)&(count[i]));
    }

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

    //calculate the size of output file
    int size_of_output_file=(sizeof(getpid())+strlen(argv[1]) +5 /*for commas and newline*/+sizeof(total_number));
    char buffer_output[size_of_output_file];

```

Στο πάνω μέρος δηλώνονται οι συναρτήσεις οι οποίες χρησιμοποιούνται στην main. Αυτό έγινε με σκοπό να υλοποιήσω πρώτα την main και μετά αυτές και το πρόγραμμα να θεωρεί ότι παρακάτω υπάρχει η υλοποίηση, επομένως δεν θα το “χτυπά”.

Ξεκινώντας την main δημιουργείται ένας πίνακας για τα thread και ένας counter τύπου data (που περιέχει δηλαδή τις πληροφορίες που θέσαμε πιο πάνω στο struct. Χρησιμοποιώντας μια δομή επανάληψης “for” βάζουμε ως πληροφορίες τους χαρακτήρες από το αρχείο που λάβαμε και τον αριθμό των thread και δημιουργούμε τα threads με την χρήση της εντολής pthread_create(). Στην οποία καλούμε την counter_words_of_file συνάρτηση για να μας υπολογίζει τις λέξεις.

Στην συνέχεια χρησιμοποιείται η pthread_join με την οποία επιτυγχάνεται το να περιμένει το ένα νήμα το άλλο για να ολοκληρωθεί πριν γράψουμε στο αρχείο. Μόλις γίνει αυτό, δημιουργείται ένας buffer για το output file για να περάσουμε τα δεδομένα.

Να σημειωθεί πως το size του output.txt υπολογίζεται αθροιστικά από το μέγεθος του pid που θα γράψουμε (υπολογίζεται με το sizeof() αφού είναι τύπου int), το μέγεθος του String του ονόματος (./directory/namefile, το οποίο επειδή είναι string υπολογίζεται με την βοήθεια της strlen()), από το μέγεθος του total_number (με την sizeof()) και τέλος από τους χαρακτήρες που προστέθηκαν όπως τα κόμματα, τα κενά και το newline.

```

int fd;
/*
permissions is like
r-- ->4          so i want 644 to give rw--w--w-
-w- ->2
--x ->1
*/
//create an output file and write on it... if already exists append that
fd=open("output.txt",O_WRONLY | O_CREAT | O_APPEND,0644 );
if (fd==-1)
{
    perror("open");
    exit(-1);
}

//prepare the format of text we will print on output file
sprintf(buffer_output,"%d, %s, %d\n",getpid(),argv[1],total_number);

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

return 0;

```

Σχετικά με την έξοδο σε αρχείο που βλέπουμε στο παραπάνω screen, δημιουργείται ένας file distributor (fd) στον οποίο περνάμε τι επιστρέφει η open(). Χρησιμοποιείται η open δηλαδή για να δημιουργήσουμε δίνοντας δικαιώματα (rw--w--w-) ή να ανοίξουμε εφόσον ήδη υπάρχει και να κάνουμε append (να γράψουμε στα ήδη υπάρχον) προκειμένου να επιτυγχάνεται η μεταφορά στο αρχείο.

Αν η open επιστρέψει αρνητική τιμή (-1) τότε κάτι πήγε λάθος και χρησιμοποιείται η perror() για να βγάλει το μήνυμα λάθους και γίνεται exit απο το πρόγραμμα.

Εφόσον όλα πάνε καλά και το fd δεν είναι ίσο με το -1, χρησιμοποιείται η sprintf() προκειμένου να περάσουμε τα δεδομένα στον buffer για την έξοδο που είχαμε δημιουργήσει και είχαμε αναφερθεί παραπάνω.

Συγκεκριμένα, η sprintf δέχεται ως ορίσματα τον buffer που θέλουμε να βάλουμε τα δεδομένα, και το string που θέλουμε να περάσουμε στο οποίο περνάμε τις τιμές του PID, του ονόματός του φακέλου και τον αριθμό των λέξεων με την εξής μορφή:

<pid>, <filename measured>, <number of words>

```

24804, ./files/fileFour.txt, 44
24795, ./files/fileTwo.txt, 451
24805, ./files/file.txt, 3

```

```

//counts the words of a file with the help of the threads
void *counter_words_of_file(void *data_args)
{
    //take the data of the thread
    data *args;
    args =data_args;

    //take the number of threads
    int number_of_threads=args->thread_counter;

    //make a counter for words of a thread
    int counter=0;
    // int i;

    //check if is the last thread to read correctly the lasts characters in buffer
    int modulo=0;
    if(number_of_threads==NTHREADS-1 && strlen(args->buffer)%NTHREADS!=0){
        modulo=(strlen(args->buffer)%NTHREADS)-1;
    }
    for(int i=number_of_threads*(strlen(args->buffer)/NTHREADS);i<=(number_of_threads+1)*(strlen(args->buffer)/NTHREADS)+modulo;i++)
    {
        //check if it is in the end of a word
        if(!word_seperator(args->buffer[i]) && word_seperator(args->buffer[i+1]))
        {
            //add by one the counter of words
            counter++;
        }
    }

    // "freeze" threads, to control threads we have to let one thread at the time
    pthread_mutex_lock(&mutex);

    //and then let that thread to add on sub of counter
    total_number+=counter;

    //and unlock - unfreeze threads
    pthread_mutex_unlock(&mutex);

    //exit of thread
    pthread_exit (NULL);
}

```

Σχετικά με την συνάρτηση `counter_words_of_file()` που χρησιμοποιήθηκε είναι μια συνάρτηση η οποία κάνει τον υπολογισμό των λέξεων πρακτικά.

Συγκεκριμένα, παίρνει ως όρισμα τις πληροφορίες απο τα threads όπως παρατηρούμε και στο screenshot με την `pthread_create()`, τα αποθηκεύουμε σε μια άλλη μεταβλητή την οποία χρησιμοποιούμε για να πάρουμε τον αριθμό των threads.

Πειραματικά διαπιστώθηκε το πρόβλημα πως ο διαχωρισμός των thread επειδη γίνεται χρησιμοποιώντας την αναλογία μέγεθος αρχείου δια των νημάτων που χρησιμοποιούμε, μπορεί να μη διαιρείται ακριβώς και να αφήνει κάποιο υπόλοιπο. Αυτό σημαίνει και απώλεια λέξεων και πληροφορίας.

Επομένως ελέγχεται αν βρισκόμαστε στο τελευταίο thread και αν έχει αφήσει κάποιο υπόλοιπο, το οποίο πρέπει να το προσθέσουμε στην περιοχή εξέτασης του κάθε thread.

Επομένως χρησιμοποιήθηκε μια μεταβλητή modulo η οποία αρχικοποιείται με 0 (αρα καλύπτει τις “τελειες” διαιρέσεις) και λαμβάνει τιμή το υπόλοιπο αυτό μειωμένο κατά 1 για να ελεγχθεί και ο προηγούμενος χαρακτήρας μήπως υπάρξει κάποιος συνδυασμός που προκύπτει λέξη και να μην αγνοηθεί.

Αυτό γιατί για να σχηματιστεί λέξη παίρνουμε ως συνθήκη το να υπάρξει η σχέση operator και γράμματος π.χ “α β” είναι 2 λέξεις καθώς στο τέλος υπάρχει ένα \0 το οποίο το λαμβάνουμε ως operator (βλέπε σελ 15 αναλυτικά για το τι θεωρείται operator στο πρόγραμμα αυτό).

Σημεία δυσκολίας/ γενικές παρατηρήσεις

Υπήρχαν μερικά σημεία τα οποία με δυσκόλεψαν κατά την διάρκεια της εργασίας που θα ήθελα να αναφερθώ.

Ένα από αυτά είναι στον έλεγχο για το αν αυτό που δεχεται είναι αρχείο. Φαίνονται 3 περιπτώσεις, 3 προσπάθειες να γίνει αυτός ο έλεγχος όμως δυσλειτουργούσαν η έβγαζαν segmentation fault και δεν κατάφερα να υλοποιήσω αυτό το ερώτημα με αυτούς τους τρόπους, οι οποίοι είναι προγραμματιστικά καλύτεροι θεωρώ καθώς κατέληξα στο να ελέγχω αν μπορεί απευθείας να ανοίξει το αρχείο και αν όχι τότε να του λέω ότι δεν είναι μπορεί να ανοιχτεί και να προχωρώ στο επόμενο έλεγχο. Αυτό πρακτικά λειτούργησε όμως η δυσκολία και οι προσπάθειες για να φτάσω μέχρι εκεί ήταν αρκετές και θεώρησα ότι έπρεπε να αναφερθεί η δυσκολία.

Ανεπιτυχείς προσπάθειες για έλεγχο αρχείου

```
//FIRST TRY
if(dir->d_type != DT_REG)
{
    printf("\t%s: it's not a file!\n",pde->d_name);
    continue;
}
```

```
//SECOND TRY
if( access(buffer_for_name_of_file, F_OK) )
{
    printf("\t%s: it's not a file!\n",pde->d_name);
    continue;
}
```

```
// THIRD TRY
if( stat(pde->d_name,&stat_buffer) == -1)
{
    printf("\t%s: it's not a file!\n",pde->d_name);
    continue;
}
```

Ένα ακόμα σημείο δυσκολίας έγκειται στην μεταφορά των πληροφοριών στους buffers και στην εκτύπωση στο αρχείο. Αρχικά θεώρησα πως η κατάλληλη συγγενική της printf εντολή γι'αυτούς τους σκοπούς είναι η sprintf καθώς αν καταλαβαίνω την χρήση της μορφοποιεί και αποθηκεύει μια σειρά χαρακτήρων και τιμών στο buffer πίνακα με το n να δηλώνει τον μέγιστο αριθμό των λέξεων που θα εγγραφεί στον

πίνακα. Αυτό εν μέρει δούλεψε, όμως στις πολλαπλές εκτελέσεις (διαδοχικές -ή μια μετά την άλλη) μετά από πολλές προσπάθειες debug παρατήρησα πως μερικές φορές διάβαζε και έγραφε “σκουπίδια” με αποτέλεσμα να περνάει λάθος τον αριθμό των λέξεων ή και να προσθέτει μερικά “κινέζικα” σύμβολα στην έξοδο. Αυτά τα αντιμετώπισα επιτυχώς χρησιμοποιώντας μια συγγενική της, την sprintf() η οποία δεν παρουσίασε τέτοια προβλήματα.

Σε αυτά τα δύο screenshots θέλω να δείξω την δυσκολία που αντιμετώπισα σχετικά με τα παραπάνω (διάβαζε “σκουπίδια”).

```
buffer output:0Qx00 ←
size output: 32
The . is not a file
The file is not in ASCII
buffer output:0000 ←
size output: 33
buffer output:0000 ←
size output: 38
```

```
guage C.
I ignore spaces and whatever else t
so numbers like 1234567890 and characters l
(46)

.: it's not a file!
fileThree.txt
This file it's not with ASCII charac
fileFour.txt
nameBuffer ./ergasia/fileFour.txt
buff Hello my name is meletis tsesmelis. I an
guage C.
I ignore spaces and whatever else tha
so numbers like 1234567890 and characters lik
(45)

00U ←
file.txt
nameBuffer ./ergasia/file.txt
buff a geia !@#5%^&*(){}<=>?:"0123456789/\.,
```

Τέλος θα ήθελα να προσθέσω πως έγινε προσπάθεια να χρησιμοποιηθούν χαμηλού επιπέδου εντολές (για παράδειγμα όχι η fopen αλλά χρησιμοποιήθηκε η open) λόγω του ότι το μάθημα αφορά αυτό το επίπεδο. Όμως ήταν κάτι που δυσκόλεψε αρκετά την υλοποίηση αυτής της εργασίας.

Ακολουθεί ένας συνοπτικός πίνακας,

Συνοπτικός Πίνακας

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

ευχαριστώ για τον χρόνο σας!