ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΜΑΤΙΚΗΣ

Αντζέλντο Χύσκαϊ

2η Εργασία στο μάθημα **Λειτουργικά Συστήματα**

Ταύρος, 28 Ιανουαρίου 2021

Περιεχόμενα

Άσκηση 2

Κώδικας

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

- Αρχείο makefile:

```
all:
  gcc it219118.c Counter.c Counter.h -lpthread
```

- Αρχείο Counter.h

```c
#ifndef WORD_COUNTER_COUNTER_H
#define WORD_COUNTER_COUNTER_H

#include <stdio.h>
#include <dirent.h>
#include <wait.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <unistd.h>
#include <assert.h>
#include <string.h>
#include <fcntl.h>
#include <signal.h>

#define NUM_THREADS 4

//that struct contains one part of text for every thread
struct text_part {
   char *array;
   int is_not_ascii;
   int start;
   int end;
   int words_founded;
};

int count_words(char array[], int length, char filename[]);

void thread_function(struct text_part *array);

void signal_handler(int);

#endif //WORD_COUNTER_COUNTER_H
```

- Αρχείο Counter.c

```c
#include <pthread.h>
#include "Counter.h"

int count_words(char text[], int text_length, char filename[]) {

   int i, tmp;

   // divide the text length with the number of threads to share the text
   int text_length_per_thread = text_length / NUM_THREADS;

   //create an array of structs to store the data of threads
   struct text_part parts[NUM_THREADS];

   //for loop to initialize the array of structs
   for (i = 0; i < NUM_THREADS; i++) {

       parts[i].array = text;
       parts[i].start = i * text_length_per_thread;
       parts[i].end = parts[i].start + text_length_per_thread;

       if (i == NUM_THREADS - 1) {
           parts[i].end = text_length;
       }

   }

   pthread_t thread[NUM_THREADS];

   // creating threads
   for (i = 0; i < NUM_THREADS; i++) {
       thread[i] = i;
       tmp = pthread_create(&thread[i], NULL, (void *(*)(void *))
thread_function, (void *) &parts[i]);

       if (tmp != 0) {

       }
   }

   int sum_of_words = 0;
```

```
    //for loop to wait for every thread to finish and add the set of words
founded
    for (int j = 0; j < NUM_THREADS; j++) {
        pthread_join(thread[j], NULL);
        sum_of_words += parts[j].words_founded;

        if (parts[j].is_not_ascii == 1) {
            return 1;
        }
    }

    int fd;

    //open the out.txt file
    if ((fd = open("out.txt", O_WRONLY | O_CREAT | O_APPEND, 0644)) == -1) {

        perror("");
        exit(1);

    }

    //calculate the length of the string for the message
    int length_of_output = snprintf(NULL, 0, "<%d>,<%s>,<%d>\n", getpid(),
filename, sum_of_words);

    //create the array and store the message
    char output[length_of_output + 1];
    snprintf(output, length_of_output + 1, "<%d>,<%s>,<%d>\n", getpid(),
filename, sum_of_words);

    //write the message on the file
    write(fd, output, strlen(output));

    return 0;
}

void thread_function(struct text_part *array_part) {

    array_part->is_not_ascii = 0;
    array_part->words_founded = 0;

    //for loop to cross the part of text
    for (int i = array_part->start; i < array_part->end; i++) {
```

```
        //check if a character is separator or is not ascii
        if (array_part->array[i] == '\0' || array_part->array[i] == '\t' ||
array_part->array[i] == '\n' ||
            array_part->array[i] == ' ') {
            array_part->words_founded++;
        } else if (array_part->array[i] > 128 || array_part->array[i] < 0) {
            array_part->is_not_ascii = 1;
            pthread_exit(NULL);
        }

    }

    pthread_exit(NULL);

}

void signal_handler(int sig_num) {

    printf("Signal Ignored!\n");
    return;

}
```

- Αρχείο it219118.c

```
#include "Counter.h"

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

    DIR *directory;
    struct stat st;
    char *curr_dir;

    //check if give one argument
    if (argc == 2) {

        curr_dir = argv[1];


        stat(curr_dir, &st);
```

```
        //check if user's argument is not a directory
        if (!S_ISDIR(st.st_mode)) {

            printf("%s : It's not a directory!\n", curr_dir);
            exit(2);

        }

    }

    //check if user has not given argument
    if (argc == 1) {

        long length;

        //get the length of current directory path
        length = pathconf(".", _PC_PATH_MAX);
        assert(length != -1);

        // allocate memory for the string
        curr_dir = malloc(length * sizeof(*curr_dir));
        assert(curr_dir);

        // get the full name of the current path
        if (getcwd(curr_dir, length) == NULL) {
            perror("getcwd");
            exit(EXIT_FAILURE);
        }

    }

    // open the directory
    directory = opendir(curr_dir);

    struct dirent *file;

    // check if the directory open properly
    if (directory == NULL) {

        printf("Directory %s cant open!\n", argv[1]);
    }


    //while loop for all files in that directory
```

```
   while ((file = readdir(directory)) != NULL) {

       //check if filename is current directory or parent directory
       if (strcmp(file->d_name, ".") == 0 || strcmp(file->d_name, "..") == 0)
{
           continue;
       }

       //calculate the length of the string and store the absolute pathname
       int length_of_filepath = snprintf(NULL, 0, "%s/%s", curr_dir,
file->d_name);
       char filepath[length_of_filepath + 1];
       snprintf(filepath, length_of_filepath + 1, "%s/%s", curr_dir,
file->d_name);

       //create the struct stat because we need information about the file
       stat(filepath, &st);

       //check if is a directory
       if (S_ISDIR(st.st_mode)) {
           printf("%s :is a directory!\n", file->d_name);
           continue;
       }

       //check if the file is empty
       if (st.st_size == 0) {
           printf("%s :File is empty!\n", file->d_name);
           continue;
       }

       //open the file stream
       int file_descriptor = open(filepath, O_RDONLY);

       //check if file has open
       if (file_descriptor == -1) {
           printf("Cannot open file!\n");
           // exit(1);
       }

       //calculate the size of array
       int text_length = st.st_size / (int) (sizeof(char));

       //create a character array and store all the characters of the file
       char text[text_length];
       int read_descriptor = read(file_descriptor, text, text_length);
```

```
        //check if the read() function has read the characters
        if (read_descriptor == -1) {
            printf("%s :Error reading file!", file->d_name);
            continue;
        }

        //initialize to check the first 10 characters of the file
        int flag = 0;
        int max_check = 10;

        //if the file has less than 10 characters
        if (text_length < max_check) {
            max_check = text_length;
        }

        // if one of the characters is not ASCII raise the flag and print the
message on the screen
        for (int i = 0; i < max_check; i++) {
            if (text[i] > 127 || text[i] < 0) {
                printf("%s :File is not ascii\n", file->d_name);
                flag = 1;
                break;
            }
        }

        //if the flag is raised then continue to the next file
        if (flag == 1) {
            continue;
        }

        //create a new child process
        int pid = fork();

        //check if the new process has created
        if (pid == -1) {
            perror("fork!");
            exit(1);
        }

        // if we are on the child process with pid 0 then call the function
count_words()
        if (pid == 0) {

            //check if count_words() has found not ASCII characters
```

```
            if (count_words(text, text_length, file->d_name) == 1) {
                printf("%s :is not ascii!\n", file->d_name);
                return 1;
            }

            return 0;

        } else { // else if we are on the parent process then ignore the
signals SIGINT and SIGTERM

            signal(SIGINT, signal_handler);
            signal(SIGTERM, signal_handler);
        }
    }

    //while loop to wait for all children
    while ((wait(NULL)) > 0);

    printf("All processes have finished!\nCheck file out.txt!\n");

    closedir(directory);

    return 0;

}
```

Τρόπος Εκτέλεσης

Για να κάνουμε compile απλώς πάμε στον φάκελο που βρίσκεται το πρόγραμμα και δίνουμε την εντολή make από το terminal. Μετά θα έχει δημιουργηθεί το αρχείο a.out και για να το εκτελέσουμε δίνουμε την εντολή ./a.out και argument τον φάκελο που βρίσκονται τα αρχεία που θέλουμε να μετρήσει τις λέξεις.

Ενδεικτικές εκτελέσεις (screenshots):

- ❏ *Βασική εκτέλεση του προγράμματος*

```
bi11@bi11-HP:~/Desktop/bi11/VSCode/Word-Counter$ make
gcc it219118.c Counter.c Counter.h -lpthread
```

```
bi11@bi11-HP:~/Desktop/bi11/VSCode/Word-Counter$ ./a.out /home/bi11/Desktop/Test
empty.txt :File is empty!
test.txt :is not ascii!
Counter.h.gch :File is not ascii
New text file :File is empty!
NewFolder :is a directory!
All processes have finished!
Check file out.txt!
```

```
bi11@bi11-HP:~/Desktop/bi11/VSCode/Word-Counter$ ./a.out
a.out :is not ascii!
Counter.h.gch :File is not ascii
.git :is a directory!
All processes have finished!
Check file out.txt!
```

```
<34879>,<five characters.txt>,<1>
<34892>,<Counter.h>,<87>
<34886>,<it219118.c>,<941>
<34880>,<README.md>,<8>
<34882>,<makefile>,<7>
```

Παρατηρήσεις/σχόλια

Δίνουμε την εντολή make για να κάνουμε compile το πρόγραμμα και τρέχουμε το αρχείο a.out είτε με παράμετρο έναν φάκελο είτε χωρίς (χωρίς παράμετρο το πρόγραμμα θα διαβάσει τα αρχεία του τρέχοντος φακέλου). Στην συνέχεια θα γράψει στο αρχείο out.txt το pid της διεργασίας που μέτρησε το κάθε αρχείο, το όνομα του αρχείου και τις λέξεις του αρχείου που κατάφερε να μετρήσει. Τα αρχεία που δεν περιέχουν χαρακτήρες ascii από 0 έως 127 , είναι κενά ή είναι φάκελοι δεν συμπεριλαμβάνονται στο αρχείο out.txt αλλά τυπώνονται στον χρήστη.

- ❏ *Δημιουργία νέων διεργασιών και φυσιολογικός τερματισμός τους*

```
        //create a new child process
        int pid = fork();

        //check if the new process has created
        if (pid == -1) {
            perror("fork!");
            exit(1);
        }

        // if we are on the child process with pid 0 then call the function
count_words()
        if (pid == 0) {

            //check if count_words() has found not ASCII characters
            if (count_words(text, text_length, file->d_name) == 1) {
                printf("%s :is not ascii!\n", file->d_name);
                return 1;
            }

            return 0;

        } else { // else if we are on the parent process then ignore the
```

Παρατηρήσεις/σχόλια

Η δημιουργία των διεργασιών γίνεται με την χρήση της συνάρτησης fork() η οποία "κλωνοποιεί" την βασική διεργασία σε δύο ολόιδιες διεργασίες με την μόνη διαφορά το pid και την τιμή που επιστρέφει η συνάρτηση fork() η οποία αποθηκεύεται στην μεταβλητή pid. Έτσι χρησιμοποιούμε αυτη την διαφοροποίηση για να αναγκάσουν τις διεργασίες να κάνουν διαφορετικά πράγματα όπως δηλαδή η εργασία "παιδί" που δημιουργείται να καλέσει την συνάρτηση count_words() και να τερματήσει ενώ η διεργασία "πατέρας" να συνεχίσει την λούπα. Η διεργασία"πατέρας" πρώτα περιμένει να τελειώσουν όλα τα παιδιά της και μετά τερματίζει ενώ οι διεργασίες "παιδια" μόλις εκτελέσουν την συνάρτηση word_counter() τερματίζουν κάνοντας απλά return.

- ❏ *Δημιουργία νημάτων και φυσιολογικός τερματισμός τους*

```
pthread_t thread[NUM_THREADS];

// creating threads
for (i = 0; i < NUM_THREADS; i++) {
    thread[i] = i;
    tmp = pthread_create(&thread[i], attr: NULL, (void *(*)(void *)) thread_function, (void *) &parts[i]);

    if (tmp != 0) {

    }
}

int sum_of_words = 0;

//for loop to wait for every thread to finish and add the set of words founded
for (int j = 0; j < NUM_THREADS; j++) {
    pthread_join(thread[j], thread_return: NULL);
    sum_of_words += parts[j].words_founded;

    if (parts[j].is_not_ascii == 1) {
        return 1;
    }
}
```

```c
void thread_function(struct text_part *array_part) {

    array_part->is_not_ascii = 0;
    array_part->words_founded = 0;

    //for loop to cross the part of text
    for (int i = array_part->start; i < array_part->end; i++) {

        //check if a character is separator or is not ascii
        if (array_part->array[i] == '\0' || array_part->array[i] == '\t' || array_part->array[i] == '\n' ||
            array_part->array[i] == ' ') {
            array_part->words_founded++;
        } else if (array_part->array[i] > 128 || array_part->array[i] < 0) {
            array_part->is_not_ascii = 1;
            pthread_exit( retval: NULL);
        }

    }

    pthread_exit( retval: NULL);

}
```

Παρατηρήσεις/σχόλια

Πρώτα δημιουργώ έναν πίνακα με pthread_t και έναν με struct array_part (περιέχει τα δεδομένα του κάθε νήματος) με τόσες θέσεις όσα είναι τα threads που θέλω να χρησιμοποιήσω. Στην συνέχεια καλώ την pthread_crate() για να δημιουργήσω τα νήματα. Ο τερματισμός των νημάτων γίνεται με την pthread_exit() αφού έχουν κάνει τις κατάλληλες αλλαγές στην μεταβλητή words_founded του struct array_part. Τέλος καλώ την pthread_join() τόσες φορές όσο είναι όλα τα νήματα που χρησιμοποιώ για να πάρω τα δεδομένα από το struct τους, επίσης δεν χρησιμοποιώ mutex διότι το κάθε νήμα κάνει αλλαγές στο δικό του struct και έτσι δεν υπάρχει περίπτωση να χρησιμοποιήσουν δύο νήματα τα ίδια δεδομένα την ίδια χρονική στιγμή.

Γενικά Σχόλια/Παρατηρήσεις

Ο τρόπος που ελέγχω αν ένα αρχείο είναι ascii γίνεται σε δύο στάδια. Το πρώτο στάδιο γίνεται στην βασική διεργασία η οποία ελέγχει τους πρώτους δέκα χαρακτήρες αν είναι ascii από 0 έως 128. Το δεύτερο γίνεται στις διεργασίες παιδιά συγκρίνοντας όλα τα γράμματα την ώρα που μετράνε τις λέξεις. Ο λόγος που το κάνω αυτό είναι επειδή δεν θέλω να επιβαρύνω τόσο πολύ την βασική εργασία που χρησιμοποιεί μόνο ένα νήμα γιατί θα αργεί αρκετά ενώ τα παιδιά της θα τελειώνουν πολύ πιο γρήγορα.

Με δυσκόλεψε / δεν υλοποίησα

Με δυσκόλεψε αρκετά το πως να διαβάσω απο τα argument το μονοπάτι του φακέλου και να διαβάσω ένα-ένα τα αρχεία που έχει μέσα, ενώ αντιθέτως αν ο χρήστης δεν έδινε argument ήταν αρκετά εύκολο.

Συνοπτικός Πίνακας

2η Εργασία		
Λειτουργία	Υλοποιήθηκε **(ΝΑΙ/ΟΧΙ/ΜΕΡΙΚΩΣ)**	**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**	