ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΜΑΤΙΚΗΣ

Νικόλαος Γουρνάκης

it22023

2η Εργασία στο μάθημα **Λειτουργικά Συστήματα**

Ταύρος, ΧΧ Ιανουαρίου 2022

Περιεχόμενα

Άσκηση 2

Κώδικας

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <pwd.h>
#include <pthread.h>

#define MAX_LINE_LENGTH 1024 * 2

// Global variables
char u_input[MAX_LINE_LENGTH];
char current_working_directory[MAX_LINE_LENGTH];
char current_user[MAX_LINE_LENGTH];
char home_dir[MAX_LINE_LENGTH] = "/home/";
FILE *log_file;
int pattern_count = 0;
pthread_mutex_t pattern_count_mutex;
// end

typedef struct
```

```
{
    char *line;
    char *pattern;

} thread_data;


void update_current_user()
{
    // Get current user
    struct passwd *pw = getpwuid(getuid());
    char *user_name = pw->pw_name;

    // Can't run as root
    if (strcmp(user_name, "root") == 0)
    {
        printf("Can't run this program as root.\n");
        exit(0);
    }

    // Update current user
    strcpy(current_user, user_name);
}

void update_current_working_dir()
{
    // Get current working directory
    char *pwd = getcwd(current_working_directory,
sizeof(current_working_directory));
    if (pwd == NULL)
    {
        perror("getcwd error");
        exit(1);
    }
}
```

```c
void log_action()
{
    // Get current time
    time_t t = time(NULL);
    struct tm tm = *localtime(&t);

    // Log the time and command to the log file
    fprintf(log_file, "%d:%d:%d = %s\n", tm.tm_hour, tm.tm_min,
tm.tm_sec, u_input);
    fflush(log_file);
}

void get_u_input()
{
    // fill u_input with '\0'
    memset(u_input, '\0', MAX_LINE_LENGTH);

    // read user input
    fgets(u_input, MAX_LINE_LENGTH, stdin);

    // remove newline character
    u_input[strlen(u_input) - 1] = '\0';

    // !FOR DEBUGGING!
    //  printf("!%s!\n", u_input);
}

void print_nav()
{
    // Update current working directory and user variables
    update_current_user();
    update_current_working_dir();

    // Print current working directory and user in bash style
    printf("%s:%s> ", current_user, current_working_directory);
}
```

```c
void *find_match(void *arg)
{
    thread_data *data = (thread_data* )arg;

    // !FOR DEBUGGING!
    // printf("%s\n", (char* ) data->line);
    // printf("%s\n", (char* ) data->pattern);

    // Find the pattern in the line
    if (strstr(data->line, data->pattern) != NULL)
    {
        printf("%s\n", data->line);

        pthread_mutex_lock(&pattern_count_mutex);
        pattern_count++;
        pthread_mutex_unlock(&pattern_count_mutex);
    }

    pthread_exit(NULL);
}

int cd(char *path)
{
    // Change directory
    if (chdir(path) != 0)
    {
        printf("ERROR: No file or directory named `%s`\n", path);
        return -1;
    }

    update_current_working_dir();
    return 0;
}
```

```c
bool run_command()
{
    // initialize status variable
    int status;

    char *command = strtok(u_input, " ");

    // Check if the command is empty
    if (command == NULL)
    {
        printf("ERROR: No command specified\n");
        return true;
    }

    // Initialize the arguments
    char* args[30];
    args[0] = command;
    int i = 1;
    while (true)
    {
        // Get the next argument
        char* token = strtok(NULL, " ");
        if (token == NULL)
        {
            // No more arguments
            args[i] = NULL;
            break;
        }
        // Add the argument to the array
        args[i] = token;
        i++;
    }

    int pid = fork();
    // Child process
    if (pid == 0)
```

```
    {
        // Execute the command
        execvp(command, args);

        // If the command is not found
        fprintf(stderr, "ERROR: Command not found `%s`\n", command);
        exit(255);
    }
    // Parent process
    else if (pid > 0)
    {
        // Wait for the child process to finish
        wait(&status);


        if (WIFEXITED(status))
        {
            // Print the exit status
            printf("Exit status: %d\n", WEXITSTATUS(status));
        }
    }
    // Error
    else
    {
        perror("fork");
        exit(1);
    }

    return false;
}

bool mygrep()
{
    // discard var is used to discard the first word of the input
    char *discard = strtok(u_input, " ");
```

```
    // Get the pattern
    char *pattern = strtok(NULL, " ");
    if (pattern == NULL)
    {
        fprintf(stderr, "ERROR: No pattern specified\n");
        return true;
    }
    // printf("%s\n", pattern);

    // Get the file path
    char *path = strtok(NULL, " ");
    if (path == NULL)
    {
        fprintf(stderr, "ERROR: No path specified\n");
        return true;
    }
    // printf("%s\n", path);

    // Open the file
    FILE *fp = fopen(path, "r");
    if (fp == NULL)
    {
        perror("ERROR");
        return true;
    }

    int num_of_lines = 1;
    char ch;
    // Count the number of lines in the file
    do
    {
        ch = fgetc(fp);

        // !FOR DEBUGGING!
        // printf("%c", ch);
```

```
        if (ch == '\n')
        {
            num_of_lines++;
        }

    } while (ch != EOF);

    // !FOR DEBUGGING!
    // printf("\n");
    // printf("%d\n", num_of_lines);
    // /\ this is 0 based

    // Create a buffer to store the lines
    fseek(fp, 0, SEEK_SET);
    char lines[num_of_lines][MAX_LINE_LENGTH];

    // Initialize the buffer
    for (int i = 0; i < num_of_lines; i++)
    {
        memset(lines[i], '\0', MAX_LINE_LENGTH);
    }

    // Read the lines into the buffer !DOES NOT SUPPORT CARRIAGE RETURNS!
    int i = 0;
    do
    {
        ch = fgetc(fp);

        // !FOR DEBUGGING!
        printf("%c", ch);

        if (ch == '\n' || ch == EOF)
        {
            strcat(lines[i], "\0");
            i++;
        }
```

```
        else
        {
            strncat(lines[i], &ch, 1);
        }
    } while (ch != EOF);

    // !FOR DEBUGGING!
    // printf("\n");
    // for(int i = 0; i < num_of_lines; i++)
    // {
    //         printf("~%s~", lines[i]);
    //         printf("\n");
    // }
    // printf("\n");

    pattern_count = 0;
    pthread_t threads[num_of_lines];
    thread_data thread_data_array[num_of_lines];

    printf("\n");

    // Create a thread for each line
    for (int i = 0; i < num_of_lines; i++)
    {
        thread_data_array[i].line = lines[i];
        thread_data_array[i].pattern = pattern;
        pthread_create(&threads[i], NULL, &find_match, (void*
)&thread_data_array[i]);
    }

    // Wait for all the threads to finish
    for (int i = 0; i < num_of_lines; i++)
    {
        pthread_join(threads[i], NULL);
    }
```

```
    // Print the number of matches
    printf("\n");
    printf("[%d matches found]\n", pattern_count);

    fclose(fp);
    return false;
}

void init()
{
    update_current_user();

    // Get the current working directory and create the log file
    update_current_working_dir();
    log_file = fopen(".myshlog", "w");
    if (log_file == NULL)
    {
        printf("Error opening log file!\n");
        exit(1);
    }

    // Get the home directory and change to it
    strcat(home_dir, current_user);
    chdir(home_dir);
    update_current_working_dir();

    // Mutex initialization
    pthread_mutex_init(&pattern_count_mutex, NULL);
}

int main()
{

    init();
    while (true)
    {
```

```c
// print the prompt and get the input and log it
print_nav();
get_u_input();
log_action();

// Exit the program
if (strcmp(u_input, "exit") == 0)
{
    break;
}
// Change the directory
else if (strncmp(u_input, "cd ", 3) == 0)
{
    strtok(u_input, " ");

    char *path = strtok(NULL, " ");
    if (path == NULL)
    {
        printf("ERROR: No path specified\n");
        continue;
    }
    cd(path);
}
// Mygrep
else if (strncmp(u_input, "mygrep", 6) == 0)
{
    bool failed = mygrep(u_input);
    if (failed)
    {
        continue;
    }
}
// Linux/Unknown Command
else
{
```

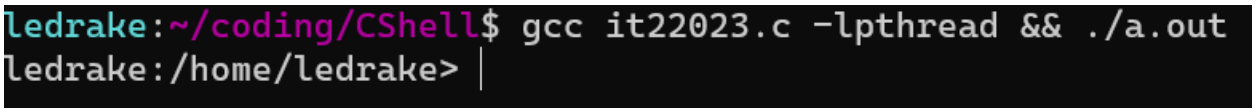
```
            bool failed = run_command(u_input);
            if (failed)
            {
                continue;
            }
        }
    }

    // Close everything and exit
    pthread_mutex_destroy(&pattern_count_mutex);
    fclose(log_file);

    return 0;
}
```

Τρόπος Εκτέλεσης

Τρέχω το προγραμμα με αυτην την γραμμη στο terminal : gcc it22023.c -lpthread && ./a.out

Ενδεικτικές εκτελέσεις (screenshots):

- *Βασική εκτέλεση του προγράμματος και εμφάνιση prompt*



- *Υποστήριξη log*

```
ledrake:~/coding/CShell$ gcc it22023.c -lpthread && ./a.out
ledrake:/home/ledrake> ls -la coding/CShell
total 68
drwxr-xr-x 3 ledrake ledrake  4096 Jan 27 22:14 .
drwxr-xr-x 4 ledrake ledrake  4096 Jan 27 19:51 ..
drwxr-xr-x 8 ledrake ledrake  4096 Jan 27 22:08 .git
-rw-r--r-- 1 ledrake ledrake    11 Jan 27 15:59 .gitignore
-rw-r--r-- 1 ledrake ledrake    31 Jan 27 22:15 .myshlog
-rw-r--r-- 1 ledrake ledrake   137 Jan 27 15:59 README.md
-rwxr-xr-x 1 ledrake ledrake 25240 Jan 27 22:14 a.out
-rw-r--r-- 1 ledrake ledrake  8232 Jan 27 22:12 it22023.c
-rwxr-xr-x 1 ledrake ledrake    34 Jan 27 22:13 run
Exit status: 0
ledrake:/home/ledrake> cd coding
ledrake:/home/ledrake/coding> cd CShell
ledrake:/home/ledrake/coding/CShell> cat run
gcc it22023.c -lpthread && ./a.outExit status: 0
ledrake:/home/ledrake/coding/CShell> exit
ledrake:~/coding/CShell$ cat .myshlog
22:15:8 = ls -la coding/CShell
22:15:20 = cd coding
22:15:24 = cd CShell
22:15:34 = cat run
22:15:38 = exit
ledrake:~/coding/CShell$
```

- *Δημιουργία νημάτων και φυσιολογικός τερματισμός τους*

```c
// Create a thread for each line
for (int i = 0; i < num_of_lines; i++)
{
    thread_data_array[i].line = lines[i];
    thread_data_array[i].pattern = pattern;
    pthread_create(&threads[i], NULL, &find_match, (void* )&thread_data_array[i]);
}

// Wait for all the threads to finish
for (int i = 0; i < num_of_lines; i++)
{
    pthread_join(threads[i], NULL);
}
```

```c
void *find_match(void *arg)
{
    thread_data *data = (thread_data* )arg;

    // !FOR DEBUGING!
    // printf("%s\n", (char* ) data->line);
    // printf("%s\n", (char* ) data->pattern);

    // Find the pattern in the line
    if (strstr(data->line, data->pattern) != NULL)
    {
        printf("%s\n", data->line);

        pthread_mutex_lock(&pattern_count_mutex);
        pattern_count++;
        pthread_mutex_unlock(&pattern_count_mutex);
    }

    pthread_exit(NULL);
}
```

- *Διαχείριση σημάτων*

  Παρατηρήσεις/σχόλια

  Δεν το εχω υλοποίηση.

- *Δημιουργία νέων διεργασιών και φυσιολογικός τερματισμός τους*

```
163     int pid = fork();
164     // Child process
165     if (pid == 0)
166     {
167         // Execute the command
168         execvp(command, args);
169 
170         // If the command is not found
171         fprintf(stderr, "ERROR: Command not found `%s`\n", command);
172         exit(255);
173     }
174     // Parent process
175     else if (pid > 0)
176     {
177         // Wait for the child process to finish
178         wait(&status);
179 
180 
181         if (WIFEXITED(status))
182         {
183             // Print the exit status
184             printf("Exit status: %d\n", WEXITSTATUS(status));
185         }
186     }
187     // Error
188     else
189     {
190         perror("fork");
191         exit(1);
192     }
193 
```

  Παρατηρήσεις/σχόλια

  Το error checking για την fork ειναι μαζι με το pid if.

- *Υλοποίηση της mygreq, δημιουργία νημάτων και φυσιολογικός τερματισμός τους*

```
// Create a thread for each line
for (int i = 0; i < num_of_lines; i++)
{
    thread_data_array[i].line = lines[i];
    thread_data_array[i].pattern = pattern;
    pthread_create(&threads[i], NULL, &find_match, (void* )&thread_data_array[i]);
}

// Wait for all the threads to finish
for (int i = 0; i < num_of_lines; i++)
{
    pthread_join(threads[i], NULL);
}
```

```c
void *find_match(void *arg)
{
    thread_data *data = (thread_data* )arg;

    // !FOR DEBUGING!
    // printf("%s\n", (char* ) data->line);
    // printf("%s\n", (char* ) data->pattern);

    // Find the pattern in the line
    if (strstr(data->line, data->pattern) != NULL)
    {
        printf("%s\n", data->line);

        pthread_mutex_lock(&pattern_count_mutex);
        pattern_count++;
        pthread_mutex_unlock(&pattern_count_mutex);
    }

    pthread_exit(NULL);
}
```

```
ledrake:/home/ledrake/coding/CShell> mygrep char it22023.c

char u_input[MAX_LINE_LENGTH];
char current_working_directory[MAX_LINE_LENGTH];
char current_user[MAX_LINE_LENGTH];
char home_dir[MAX_LINE_LENGTH] = "/home/";
    char *line;
    char *pattern;
    char *user_name = pw->pw_name;
    char *pwd = getcwd(current_working_directory, sizeof(current_working_directory));
    // remove newline character
    // printf("%s\n", (char* ) data->line);
    // printf("%s\n", (char* ) data->pattern);
int cd(char *path)
    char *command = strtok(u_input, " ");
    char* args[30];
        char* token = strtok(NULL, " ");
    char *discard = strtok(u_input, " ");
    char *pattern = strtok(NULL, " ");
    char *path = strtok(NULL, " ");
    char ch;
    char lines[num_of_lines][MAX_LINE_LENGTH];
            char *path = strtok(NULL, " ");

[21 matches found]
ledrake:/home/ledrake/coding/CShell> exit
ledrake:~/coding/CShell$
```

Παρατηρήσεις/σχόλια

Δεν υποστηρίζει carriage return / “\r”.

- ❏ *Ομαλή εκτέλεση προγράμματος, error handling, τεκμηρίωση*

```
ledrake:~/coding/CShell$ gcc it22023.c -lpthread && ./a.out
ledrake:/home/ledrake> ls -la coding/CShell
total 68
drwxr-xr-x 3 ledrake ledrake  4096 Jan 27 22:14 .
drwxr-xr-x 4 ledrake ledrake  4096 Jan 27 19:51 ..
drwxr-xr-x 8 ledrake ledrake  4096 Jan 27 22:08 .git
-rw-r--r-- 1 ledrake ledrake    11 Jan 27 15:59 .gitignore
-rw-r--r-- 1 ledrake ledrake    31 Jan 27 22:15 .myshlog
-rw-r--r-- 1 ledrake ledrake   137 Jan 27 15:59 README.md
-rwxr-xr-x 1 ledrake ledrake 25240 Jan 27 22:14 a.out
-rw-r--r-- 1 ledrake ledrake  8232 Jan 27 22:12 it22023.c
-rwxr-xr-x 1 ledrake ledrake    34 Jan 27 22:13 run
Exit status: 0
ledrake:/home/ledrake> cd coding
ledrake:/home/ledrake/coding> cd CShell
ledrake:/home/ledrake/coding/CShell> cat run
gcc it22023.c -lpthread && ./a.outExit status: 0
```

```
ledrake:/home/ledrake/coding/CShell> mygrep char it22023.c

char u_input[MAX_LINE_LENGTH];
char current_working_directory[MAX_LINE_LENGTH];
char current_user[MAX_LINE_LENGTH];
char home_dir[MAX_LINE_LENGTH] = "/home/";
    char *line;
    char *pattern;
    char *user_name = pw->pw_name;
    char *pwd = getcwd(current_working_directory, sizeof(current_working_directory));
    // remove newline character
    // printf("%s\n", (char* ) data->line);
    // printf("%s\n", (char* ) data->pattern);
int cd(char *path)
    char *command = strtok(u_input, " ");
    char* args[30];
        char* token = strtok(NULL, " ");
    char *discard = strtok(u_input, " ");
    char *pattern = strtok(NULL, " ");
    char *path = strtok(NULL, " ");
    char ch;
    char lines[num_of_lines][MAX_LINE_LENGTH];
            char *path = strtok(NULL, " ");

[21 matches found]
ledrake:/home/ledrake/coding/CShell> exit
ledrake:~/coding/CShell$
```

```
ledrake:/home/ledrake> cd not_a_dir
ERROR: No file or directory named `not_a_dir`
ledrake:/home/ledrake> mygrep
ERROR: No pattern specified
ledrake:/home/ledrake> mygrep pattern
ERROR: No path specified
ledrake:/home/ledrake> mygrep pattern not_a_file
ERROR: No such file or directory
```

```
ledrake:/home/ledrake> mygrep pattern coding/CShell/file_without_read_permisions
ERROR: Permission denied
ledrake:/home/ledrake> not_a_command
ERROR: Command not found `not_a_command`
Exit status: 255
```

```
ledrake:~/coding/CShell$ gcc it22023.c -lpthread
ledrake:~/coding/CShell$ sudo ./a.out
Can't run this program as root.
```

Παρατηρήσεις/σχόλια

Το προγραμμα δεν μπορει να τρεξει σαν root

Γενικά Σχόλια/Παρατηρήσεις

Κανω την εργασια σε windows μεσω του WSL , και λογω αυτο ο editor μου ειναι σε Windows mode αρα βαζει CRLF που προκαλεσε ενα αορατο bug στην mygrep που μου πηρε 8 ωρες για να το βρω, επισης δεν χρησιμοποιω system calls οπως open,write,read καθως προκαλουν απροσδιοριστες καταστασεις οποτε πηγα πισω σε αυτα που ξερω fopen,fprintf,fputs.

Με δυσκόλεψε / δεν υλοποίησα

Το signal handling , το πως δουλεύει το prompt μου ειναι τυπώνει το user:current_dir> και περιμενει για τον χρηστη να γραψει κατι , αυτο επιτυγχανεται με μια fgets. Αμα οταν περιμενει το

προγραμμα για user input και στειλει signal TERM | INT , γινεται handle το signal αλλα μετα δεν ηξερα πως να κανω restart το loop καθως μετα απο το σημα γυρναει στο fgets και ειναι ασχημο.

Συνοπτικός Πίνακας

2η Εργασία		
Λειτουργία	Υλοποιήθηκε **(ΝΑΙ/ΟΧΙ/ΜΕΡΙΚΩΣ)**	**Συνοπτικές Παρατηρήσεις**
Βασική διεργασία η οποία ελέγχει τις παραμέτρους εμφανίζει το promt στο χρήστη	**ΝΑΙ**	
Υποστήριξη log	**ΝΑΙ**	
Δημιουργία νέων διεργασιών και φυσιολογικός τερματισμός τους	**ΝΑΙ**	
Υλοποίηση της mygreq, δημιουργία νημάτων και φυσιολογικός τερματισμός τους	**ΝΑΙ**	Το carriage return/ “\r” μεσα στο αρχειο που κανουμε mygrep χαλαει ολο το mygrep.
Ομαλή εκτέλεση προγράμματος, error handling, τεκμηρίωση	**ΝΑΙ**	