



ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΜΑΤΙΚΗΣ

Γιώργος Νικολαΐδης

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

Τάυρος, 24 Ιανουαρίου 2023

Περιεχόμενα

Άσκηση 2	3
Κώδικας	3
Τρόπος Εκτέλεσης	7
Ενδεικτικές εκτελέσεις (screenshots):	7
Βασική διεργασία η οποία ελέγχει τη δημιουργία των κατάλληλων διεργασιών και τον έλεγχο του προγράμματος	7
Screenshots	7
Συγχρονισμός των 2 διεργασιών εγγραφής ανάγνωσης	8
Screenshots	8
Διαχείριση σημάτων	8
Screenshots	8
Δημιουργία νημάτων και πέρασμα παραμέτρων	9
Screenshots	9
Συγχρονισμός νημάτων και σωστή διαδικασία μέτρησης αποτελέσματος	9
Screenshots	9
Ομαλή εκτέλεση προγράμματος, error handling, τεκμηρίωση	10
Screenshots	10
Γενικά Σχόλια/Παρατηρήσεις	11
Με δυσκόλεψε / δεν υλοποίησα	11
Συνοπτικός Πίνακας	12

Άσκηση 2

Κώδικας

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

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <fcntl.h>
#include <sys/file.h>
#include <pthread.h>
#include <time.h>
#include <math.h>

#define N_THREADS 4
#define SIZE 2000

void signal_handler(int signum);
void *thread_func(void *args);
void child();
void parent();
int sum = 0;
int array[26];
pthread_mutex_t mymutex = PTHREAD_MUTEX_INITIALIZER;

main()
{
    int fd;
    // create a file
    if ((fd = open("data.txt", O_CREAT, 0666)) == -1)
    {
        perror("open");
    }

    printf("PID:%d\n", getpid());
    { // signal handling
        signal(SIGTERM, signal_handler); // handle kill-15
        signal(SIGINT, signal_handler); // handle kill-2
    }
```

```

    }
    int pid, status;
    if ((pid = fork()) == -1) // check for error using fork
    {
        perror("fork");
        exit(EXIT_FAILURE);
    }
    else if (pid != 0) // parents code
    {
        printf("Parent process ..... \n");
        parent();
        waitpid(pid, &status, WNOHANG);
    }
    else if (pid == 0) // child code
    {
        sleep(4);
        child();
    }
    sleep(5);
}

void child()
{
    {
        int fd;
        // open fd
        if ((fd = open("data.txt", O_RDONLY)) == -1)
        {
            perror("open");
            exit(EXIT_FAILURE);
        }
        printf("Pid of process PID:%d\n", getpid());
        pthread_t threads[N_THREADS];

        int count[N_THREADS];

        for (int i = 0; i < N_THREADS; i++)
        {
            count[i] = i;
            pthread_create(&threads[i], NULL, thread_func, &fd);
        }
        for (int i = 0; i < N_THREADS; i++)
        {
            pthread_join(threads[i], NULL);
        }
        for(int i=0;i<26;i++)
        {

```

```

        printf("%c appears %d times\n" , (97+i) , array[i]);
    }
    for (int i = 0; i < 26; i++)
    {
        sum += array[i];
    }

    printf("The file contains %d character from a-z\n", sum);

    close(fd);
}
}

void signal_handler(int signum) // void to handle signals
{
    sigset_t mask;

    char response;
    printf("Are you sure you want to exit?\n");
    scanf("%c", &response);
    if (response != 'Y' | 'y')
    {
        exit(signum);
    }
    else
        sigfillset(&mask);
    sigprocmask(SIG_SETMASK, &mask, NULL);
}

void *thread_func(void *args)
{
    int *fd = args;
    char buff[500];
    read(*fd, buff, 500);

    int b;
    char c;
    for(int i=0;i<500;i++)
    {
        if(buff[i]>=97 && buff[i]<=122)
        {
            pthread_mutex_lock(&mymutex);
            c = buff[i];
            b = (int) c % 97 ;
            array[b]++;
            pthread_mutex_unlock(&mymutex);
        }
    }
}

```

```
    }  
}  
  
void parent()  
{  
    { // open the file  
        srand(time(NULL));  
        int fd, bytes;  
        int buf[SIZE];  
        char buffer[SIZE];  
  
        if ((fd = open("data.txt", O_WRONLY)) == -1) // check for errors during open  
        {  
            perror("open");  
            exit(EXIT_FAILURE);  
        }  
        // write to the file  
        for (int i = 0; i < SIZE; i++)  
        {  
            buf[i] = (rand() % (122 - 97 + 1)) + 97;  
            buffer[i] = buf[i];  
        }  
        bytes = write(fd, buffer, sizeof(buffer));  
        printf("Bytes were written SIZE:%d\n", bytes);  
        close(fd);  
    }  
}
```

Τρόπος Εκτέλεσης

Το πρόγραμμα υλοποιήθηκε σε ένα αρχείο .c. Συνεπώς δεν χρειαζόμαστε makefile για να το κάνουμε compile. Με την εντολή:

```
gcc -o (executable) (filename)
```

Το κάνουμε compile και στην συνέχεια το εκτελούμε με την εντολή

```
./(executable)
```

Παρόλα αυτά για λόγους πρακτικότητας υλοποίησα και ένα makefile με εντολές

make-> για compile

make run -> για εκτέλεση

make clean -> για διαγραφή

Ενδεικτικές εκτελέσεις (screenshots):

- **Βασική διεργασία η οποία ελέγχει τη δημιουργία των κατάλληλων διεργασιών και τον έλεγχο του προγράμματος**

Screenshots

```
int pid, status;
if ((pid = fork()) == -1) // check for error using fork
{
    perror("fork");
    exit(EXIT_FAILURE);
}
else if (pid != 0) // parents code
{
    printf("Parent process .....\\n");
    parent();
    waitpid(pid, &status, WNOHANG);
}
else if (pid == 0) // child code
{
    sleep(4);
    child();
}
sleep(5);
```

- *Συγχρονισμός των 2 διεργασιών εγγραφής ανάγνωσης*

Μέσω της sleep συγχρονίζεται το πρόγραμμα και τρέχει πρώτα ο πατέρας και μετά το παιδί.

Screenshots

```
sleep(4);
child();
```

Στο τέλος για να πάρει σήμα ο πατέρας ότι τελείωσε το παιδί υλοποίησα μια waitpid

```
waitpid(pid, &status, WNOHANG);
```

- *Διαχείριση σημάτων*

Για την διαχείριση των σημάτων υλοποίησα μια signal handler η οποία καλείται μόνο για τα σήματα SIGINT και SIGSTOP

Screenshots

```
void signal_handler(int signum) // void to handle signals
{
    sigset_t mask;

    char response;
    printf("Are you sure you want to exit?\n");
    scanf("%c", &response);
    if (response != 'Y' | 'y')
    {
        exit(signum);
    }
    else
    {
        sigfillset(&mask);
        sigprocmask(SIG_SETMASK, &mask, NULL);
    }
}
```


- **Δημιουργία νημάτων και πέρασμα παραμέτρων**

Στην δημιουργία νημάτων κλήθηκε η `pthread_create` όπου πέρασα σαν παράμετρο τον file descriptor ώστε να μπορώ να κάνω read μέσω των threads.

Screenshots

```
pthread_create(&threads[i], NULL, thread_func, &fd);
```

- **Συγχρονισμός νημάτων και σωστή διαδικασία μέτρησης αποτελέσματος**

Όστε να βγαίνουν σωστά αποτελέσματα και να συγχρονίζονται τα νήματα μεταξύ τους χρησιμοποίησα mutexes όπου τους κλείδωνω πριν από κάθε πρόσθεση και τον ξεκλειδωνω αμέσως μετά.

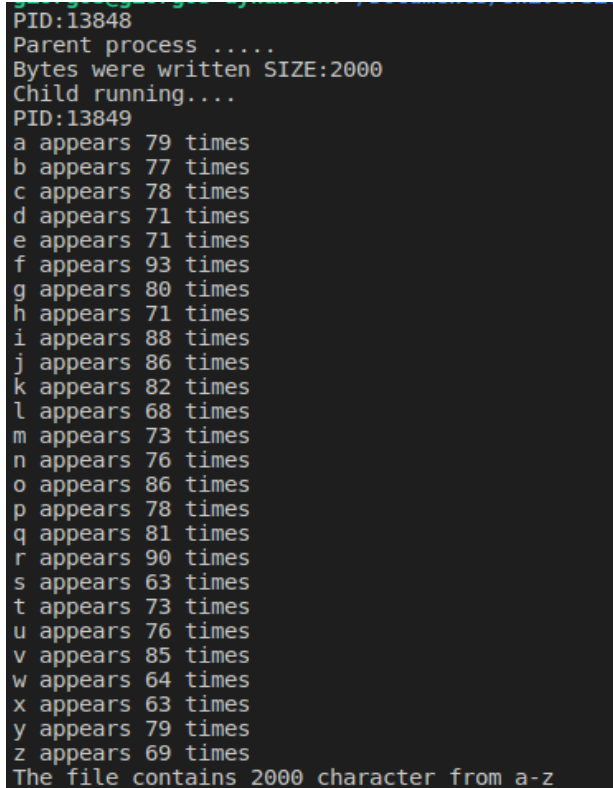
Screenshots

```
pthread_mutex_lock(&mymutex);  
c = buff[i];  
b = (int) c % 97 ;  
array[b]++;  
pthread_mutex_unlock(&mymutex);
```

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

Στο output φαίνεται το process id, πότε τρέχει ο πατέρας, πότε το παιδί, όταν γραφτούν με επιτυχία τα δεδομένα στο file και τυπώνει ποσες φορές εμφανίζεται ο καθε χαρακτήρας ξεχωριστά και το συνολικό άθροισμα των χαρακτήρων.

Screenshots



```
PID:13848
Parent process .....
Bytes were written SIZE:2000
Child running....
PID:13849
a appears 79 times
b appears 77 times
c appears 78 times
d appears 71 times
e appears 71 times
f appears 93 times
g appears 80 times
h appears 71 times
i appears 88 times
j appears 86 times
k appears 82 times
l appears 68 times
m appears 73 times
n appears 76 times
o appears 86 times
p appears 78 times
q appears 81 times
r appears 90 times
s appears 63 times
t appears 73 times
u appears 76 times
v appears 85 times
w appears 64 times
x appears 63 times
y appears 79 times
z appears 69 times
The file contains 2000 character from a-z
```

Γενικά Σχόλια/Παρατηρήσεις

Για να επιτύχει το πρόγραμμα χρειάστηκε να χρησιμοποιησω καθολικές μεταβλητές και πολλές void. Στην κάθε διεργασία αντιστοιχεί μια void. Αρχικά ξεκίνησα τυπώνοντας το pid, και στην συνέχεια δημιούργησα το file. Έπειτα έκανα fork την διεργασία, έκανα τους απαραίτητους ελέγχους επιτυχίας, και χώρισα τον κώδικα σε πατέρα και παιδί. Στη συνέχεια μέσω της parent κάλεσα την συνάρτηση write που γράφει στο αρχείο random χαρακτήρες. Σε περίπτωση επιτυχίας τυπώνει ότι γραφτηκαν με επιτυχία οι λέξεις. Όταν και εφόσον γραφτούν οι λέξεις στο αρχείο το παιδί πάει και δημιουργεί 4 threads στα οποία περνά σαν όρισμα τον file descriptor. Από εκεί διαβάζεται το περιεχόμενο του αρχείου και προστίθεται σε ένα sum το πλήθος του κάθε γράμματος απο το a-z. Τα threads ενώνονται/κανουν join ώστε να αποφευχθούν τυχόν λάθοι και τυπώνονται. Το παιδί επιστρέφει πίσω στον πατέρα που το περιμένει μέσω της waitpid και το πρόγραμμα τερματίζεται.

Με δυσκόλεψε / δεν υλοποίησα

Με δυσκόλεψε η διαδικασία να ανοίξω file διότι αρχικά δημιούργησα το file με O_CREATE | O_WRONLY και permissions 0666 όμως στην διαδικασία της open για O_RDONLY μου έβγαζε το perror "open: permission denied " και αναγκάστηκα να το υλοποιήσω με διαφορετικό τρόπο.

Συνοπτικός Πίνακας

2η Εργασία		
Λειτουργία	Υλοποιήθηκε (ΝΑΙ/ΟΧΙ/ΜΕ ΡΙΚΩΣ)	Συνοπτικές Παρατηρήσεις
Βασική διεργασία η οποία ελέγχει τη δημιουργία των κατάλληλων διεργασιών και τον έλεγχο του προγράμματος		
Συγχρονισμός των 2 διεργασιών εγγραφής ανάγνωσης		
Διαχείριση σημάτων		
Δημιουργία νημάτων και πέρασμα παραμέτρων		
Συγχρονισμός νημάτων και σωστή διαδικασία μέτρησης αποτελέσματος		
Ομαλή εκτέλεση προγράμματος, error handling, τεκμηρίωση		