ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΜΑΤΙΚΗΣ

Καρατζιάς Κυριάκος

2η Εργασία στο μάθημα **Λειτουργικά Συστήματα**

Ταύρος, 25 Ιανουαρίου 2024

Περιεχόμενα

Άσκηση 2

Κώδικας

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

```
#include <fcntl.h>
#include <pthread.h>
#include <semaphore.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <time.h>
#include <unistd.h>
```

```c
#define NTHREADS 4
#define LINES 100
#define LINE_NUMS 50
#define LINES_PER_THREAD (LINES / NTHREADS)
#define REMAINING_LINES (LINES % NTHREADS)
#define SIZE_OF_LINES (LINE_NUMS * sizeof(int) + LINE_NUMS * 1 + 1)

typedef struct {  // use this struct at thread_func
    int fd, thread, sum, lines;
} threads_arguments;

int global_sum = 0;  // Variable added all numbers
int fd;

pthread_mutex_t mutexSum = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutexRead = PTHREAD_MUTEX_INITIALIZER;

void* thread_func(void* args) {
    char eat;
    threads_arguments* ptr = args;  // getting the struct
    int local_sum, fd, thread, num_read;
    local_sum = 0;
    fd = ptr->fd;
    thread = ptr->thread;

    int buf[LINE_NUMS];

    for (int cur_line = LINES_PER_THREAD * thread;
         cur_line < (thread + 1) * LINES_PER_THREAD;
         cur_line++) {                  // for every thread for run 25 times
        pthread_mutex_lock(&mutexRead);  // Critical area for reading file

        lseek(fd, cur_line * SIZE_OF_LINES,
              SEEK_SET);  // setting the cursor to the right place

        for (int i = 0; i < LINE_NUMS; i++) {
            read(fd, &buf[i], sizeof(int));  // reading data and save in buf[]
            read(fd, &eat, 1);               // removes space
        }
        read(fd, &eat, 1);  // removes \n
        pthread_mutex_unlock(&mutexRead);

        for (int i = 0; i < LINE_NUMS; i++) {
            local_sum += buf[i];  // adding all read data from buf[] to
                                  // local_sum
            buf[i] = 0;           // reset buf
        }
        num_read++;
    }
    if (REMAINING_LINES != 0) {  // Check that all lines was read
    }
```

```
    pthread_mutex_lock(
        &mutexSum);  // adding sum to global variable...we need mutex (lock)
    global_sum += local_sum;
    pthread_mutex_unlock(&mutexSum);  //(unlock)
    ptr->sum = local_sum;             // set to struct data
    ptr->lines = num_read;
    pthread_exit(NULL);  // thread return
}

void signal_management(int sig) {
    char ch;
    if (sig == SIGINT) {
        printf("Received SIGINT signal\nDo you want to stop executing?(Y/N)");
        scanf("%c", &ch);
        if ((ch == 'Y') || (ch == 'y')) {
            exit(2);
        }
    } else if (sig == SIGTERM) {
        printf("Received SIGTERM signal\nDo you want to stop executing?(Y/N)");
        scanf("%c", &ch);
        if ((ch == 'Y') || (ch == 'y')) {
            exit(15);
        }
    }
}
int main(int argc, char** argv) {
    // Signals managment
    signal(SIGINT, signal_management);  // setting actions for signals
    signal(SIGTERM, signal_management);

    int check_counter = 0;

    // Variables for genarate the random numbers
    int randomNum;

    // Variables for the fork
    int pid;
    int status;

    // Variables for the semaphore and create semaphore
    sem_t *semA, *semB;
    const char* semName1 = "it2022120A";
    const char* semName2 = "it2022120B";
    semA = sem_open(semName1, O_CREAT | O_TRUNC, 0600, 1);  // Open semaphores
    semB = sem_open(semName2, O_CREAT | O_TRUNC, 0600, 0);

    // Variables for threads
    pthread_t threads[NTHREADS];

    pid = fork();  // Fork and check for error (creating new process, child
                   // process)
```

```
    if (pid == -1) {
        perror("fork");
        exit(1);
    }
    /*The PARENT process*/
    if (pid != 0) {
        srand(time(NULL));  // Initialize rand numbers.
        fd = open("data.txt", O_WRONLY | O_CREAT |O_TRUNC,0644);                //
Open file and check for error to open
        if (fd == -1) {
            perror("Error opening file!\n");
            exit(1);
        }
        printf("Parent process!\n");
        sem_wait(semA);  // semaphore block

        for (int i = 0; i < LINES; i++) {
            for (int j = 0; j < LINE_NUMS; j++) {
                randomNum =
                    rand() % 101;  // generating random number based in time
                check_counter += randomNum;  // keep the check_counter to check
                                             // the output later
                write(fd, &randomNum, sizeof(int));  // Writes a number
                write(fd, " ", 1);  // Add space between numbers
            }
            write(fd, "\n", 1);  // Add endline after write 50 numbers
        }
        close(fd);  // close file
        printf("Output must be: %d\n", check_counter);
        sem_post(semB);                    // unblock - set available
        waitpid(pid, &status, WUNTRACED);  // wait for the child
        sem_close(semA);                   // close semaphores
        sem_close(semB);
        sem_unlink(semName1);  // remove a named semaphore from the system
        sem_unlink(semName2); //Here the program end
    } else { /*The child process*/

        printf("Child Process!\n");
        fd = open("data.txt", O_RDONLY);
        if (fd == -1) {
            perror("Error opening file!\n");
            exit(1);
        }

        threads_arguments arg_thr[NTHREADS];  // Struct to be able to use more
                                              // variables in thread_func
        pthread_t threads[NTHREADS];
        sem_wait(semB);  // semaphore block

        for (int k = 0; k < NTHREADS; ++k) {
            arg_thr[k].fd = fd;  // setting data to struct to send them in
                                 // function thread_func
```

```
            arg_thr[k].thread = k;
            if (pthread_create(&threads[k], NULL, thread_func, &arg_thr[k]) !=
                0) {  // create threads
                perror("Error create thread");
                exit(1);
            }
        }
        sem_post(semA);                       // unblock semaphore
        for (int k = 0; k < NTHREADS; k++) {  // Wait for all threads to finish
            if (pthread_join(threads[k], NULL) !=
                0) {  // wait threads to join (come back from thread_func)
                perror("Error joining thread");
                exit(1);
            }
        }

        for (int i = 0; i < NTHREADS; i++) {  // print details for every thread
            printf("Thread: %d\nRead lines: %d\nIts local sum is: %d\n", i + 1,
                   arg_thr[i].lines, arg_thr[i].sum);
        }
        printf("The total sum for all threads is:%d\n",
               global_sum);  // print the sum of the random generated numbers
        close(fd);           // close file
    }
}
```

Τρόπος Εκτέλεσης

Ο κώδικας δεν χρειάζεται κάποιο αρχείο για την εκτέλεση. Ότι χρηστιμοποιεί το δημιουργεί από μόνο του (file: “data.txt”).

Μεταγλώτηση: gcc -lpthread it2022120.c

Εκτέλεση: ./a.out

```
karatziask$~/code/semester_3/oper_sys/anafora2 gcc -lpthread it2022120.c
karatziask$~/code/semester_3/oper_sys/anafora2 ./a.out
```

Ενδεικτικές εκτελέσεις (screenshots):

Βασική διεργασία η οποία ελέγχει τη δημιουργία των κατάλληλων διεργασιών και τον έλεγχο του προγράμματος: ***(child process)***

```
155     } else { /*The child process*/
156
157         printf("Child Process!\n");
158         fd = open("data.txt", O_RDONLY);
159         if (fd == -1) {
160             perror("Error opening file!\n");
161             exit(1);
162         }
163
164         threads_arguments arg_thr[NTHREADS];  // Struct to be able to use more
165                                               // variables in thread_func
166         pthread_t threads[NTHREADS];
167         sem_wait(semB);  // semaphore block
168
169         for (int k = 0; k < NTHREADS; ++k) {
170             arg_thr[k].fd = fd;  // setting data to struct to send them in
171                                  // function thread_func
172             arg_thr[k].thread = k;
173             if (pthread_create(&threads[k], NULL, thread_func, &arg_thr[k]) !=0 ) {  // create threads
174                 perror("Error create thread");
175                 exit(1);
176             }
177         }
178         sem_post(semA);                     // unblock semaphore
179         for (int k = 0; k < NTHREADS; k++) {  // Wait for all threads to finish
180             if (pthread_join(threads[k], NULL) !=
181                 0) {  // wait threads to join (come back from thread_func)
182                 perror("Error joining thread");
183                 exit(1);
184             }
185         }
186
187         for (int i = 0; i < NTHREADS; i++) {  // print details for every thread
188             printf("Thread: %d\nRead lines: %d\nIts local sum is: %d\n", i + 1,
189                    arg_thr[i].lines, arg_thr[i].sum);
190         }
191         printf("The total sum for all threads is:%d\n",
192                global_sum);  // print the sum of the random generated numbers
193         close(fd);           // close file
```

Συγχρονισμός των 2 διεργασιών εγγραφής ανάγνωσης: *(έχουν γίνει οι κατάλληλες αλλαγές για τα printf)*

```
karatziask$~/code/semester_3/oper_sys/anafora2 gcc -lpthread it2022120.c
karatziask$~/code/semester_3/oper_sys/anafora2 ./a.out
Parent process!
Child Process!
semA block!
Writing in file ended!
Output must be: 255133
semB unblock
semB block!
semA unblock
Thread: 1
Read lines: 25
Its local sum is: 63383
Thread: 2
Read lines: 25
Its local sum is: 62929
Thread: 3
Read lines: 25
Its local sum is: 64538
Thread: 4
Read lines: 25
Its local sum is: 64283
The total sum for all threads is:255133
karatziask$~/code/semester_3/oper_sys/anafora2
```

Διαχείριση σημάτων

```c

void signal_management(int sig) {
    char ch;
    if (sig == SIGINT) {
        printf("Received SIGINT signal\nDo you want to stop executing?(Y/N)");
        scanf("%c", &ch);
        if ((ch == 'Y') || (ch == 'y')) {
            exit(2);
        }
    } else if (sig == SIGTERM) {
        printf("Received SIGTERM signal\nDo you want to stop executing?(Y/N)");
        scanf("%c", &ch);
        if ((ch == 'Y') || (ch == 'y')) {
            exit(15);
        }
    }
}
```

```
karatziask$~/code/semester_3/oper_sys/anafora2 ./a.out
^CReceived SIGINT signal
Do you want to stop executing?(Y/N)n
Child Process!
Parent process!
Output must be: 250325
Thread: 1
Read lines: 25
Its local sum is: 63005
Thread: 2
Read lines: 25
Its local sum is: 62924
Thread: 3
Read lines: 25
Its local sum is: 63645
Thread: 4
Read lines: 25
Its local sum is: 60751
The total sum for all threads is:250325
karatziask$~/code/semester_3/oper_sys/anafora2 ./a.out
^CReceived SIGINT signal
Do you want to stop executing?(Y/N)y
karatziask$~/code/semester_3/oper_sys/anafora2
```

Δημιουργία νημάτων και πέρασμα παραμέτρων

```c
            if (pthread_create(&threads[k], NULL, thread_func, &arg_thr[k]) !=0 ) {  // create threads
```

```c
typedef struct {  // use this struct at thread_func
    int fd, thread, sum, lines;
} threads_arguments;
```

Συγχρονισμός νημάτων και σωστή διαδικασία μέτρησης αποτελέσματος

```c
void* thread_func(void* args) {
    char eat;
    threads_arguments* ptr = args;  // getting the struct
    int local_sum, fd, thread, num_read;
    local_sum = 0;
    fd = ptr->fd;
    thread = ptr->thread;

    int buf[LINE_NUMS];

    for (int cur_line = LINES_PER_THREAD * thread;
         cur_line < (thread + 1) * LINES_PER_THREAD;
         cur_line++) {                  // for every thread for run 25 times
        pthread_mutex_lock(&mutexRead);  // Critical area for reading file

        lseek(fd, cur_line * SIZE_OF_LINES,
              SEEK_SET);  // setting the cursor to the right place

        for (int i = 0; i < LINE_NUMS; i++) {
            read(fd, &buf[i], sizeof(int));  // reading data and save in buf[]
            read(fd, &eat, 1);               // removes space
        }
        read(fd, &eat, 1);  // removes \n
        pthread_mutex_unlock(&mutexRead);

        for (int i = 0; i < LINE_NUMS; i++) {
            local_sum += buf[i];  // adding all read data from buf[] to
                                  // local_sum
            buf[i] = 0;  // reset buf
        }
        num_read++;
    }
    if (REMAINING_LINES != 0) {  // Check that all lines was read
    }

    pthread_mutex_lock(
        &mutexSum);  // adding sum to global variable...we need mutex (lock)
    global_sum += local_sum;
    pthread_mutex_unlock(&mutexSum);  //(unlock)
    ptr->sum = local_sum;             // set to struct data
    ptr->lines = num_read;
    pthread_exit(NULL);  // thread return
}
```

```
Thread: 1
Read lines: 25
Its local sum is: 63005
Thread: 2
Read lines: 25
Its local sum is: 62924
Thread: 3
Read lines: 25
Its local sum is: 63645
Thread: 4
Read lines: 25
Its local sum is: 60751
The total sum for all threads is:250325
karatziask$~/code/semester_3/oper_sys/anafora2 ./a.out
```

Ομαλή εκτέλεση προγράμματος, error handling, τεκμηρίωση

```
karatziask$~/code/semester_3/oper_sys/anafora2 time ./a.out
Parent process!
Child Process!
Writing in file ended!
Output must be: 247469
Thread: 1
Read lines: 25
Its local sum is: 62604
Thread: 2
Read lines: 25
Its local sum is: 63091
Thread: 3
Read lines: 25
Its local sum is: 60350
Thread: 4
Read lines: 25
Its local sum is: 61424
The total sum for all threads is:247469

real    0m0,042s
user    0m0,016s
sys     0m0,027s
karatziask$~/code/semester_3/oper_sys/anafora2
```

Γενικά Σχόλια/Παρατηρήσεις

Parent process- γράφει τυχαίους αριθμούς στο file (data.txt).

Child process- Περιμένει την διεεγασία πατέρα να τερματίσει, δημιουργεί 4 νήματα καλώντας την συνάρτηση thread_func για να διαβάσουν τους αριθμούς απο το αρχείο και να τους προσθέσουν όλους στο local_sum.

Κάθε νήμα διαβάζει από 25 γραμές με αριθμούς.

Για την εγγραφή στην global μεταβλητή “global_sum” χρησιμοποιούμε mutex.

Συνοπτικός Πίνακας

2η Εργασία		
Λειτουργία	Υλοποιήθηκε **(ΝΑΙ/ΟΧΙ/ ΜΕΡΙΚΩΣ)**	**Συνοπτικές Παρατηρήσεις**
Βασική διεργασία η οποία ελέγχει τη δημιουργία των κατάλληλων διεργασιών και τον έλεγχο του προγράμματος	**ΝΑΙ**	Child process
Συγχρονισμός των 2 διεργασιών εγγραφής ανάγνωσης	**ΝΑΙ**	Πρώτα γίνετε η εγγραφή από την διεγασία πατέρα και μετά το παιδί αναλαμβάνει την δημιουργία νημάτων για ανάγνωση των αριθμών.
Διαχείριση σημάτων	**ΝΑΙ**	signal_management function
Δημιουργία νημάτων και πέρασμα παραμέτρων	**ΝΑΙ**	Οι παραμέτροι είναι ο file descriptor, thread (“id”-”number”),sum (παίρνει το σύνολο του κάθε thread), lines (παίρνει το σύνολο των γραμμών του κάθε thread που διαβάστηκε).
Συγχρονισμός νημάτων και σωστή διαδικασία μέτρησης αποτελέσματος	**Ναι**	Γ*ίνετε μέσα στην συνάρτηση* thread_func
Ομαλή εκτέλεση προγράμματος, error handling, τεκμηρίωση	**Ναι**	-