**Probleme rezolvate THREADS**

**11. a thread which calculates the number of digits, the number of leters and the number of special characters**

#include <stdio.h>

#include <pthread.h>

#include <string.h>

typedef struct{

pthread\_mutex\_t \*m;

int \*digits, \*letters, \*specials;

char\* string;

} data;

void\* f(void\* arg){

data d = \*((data\*) arg);

int d = 0, l = 0, s = 0;

char\* string = d.string;

int len = strlen(string);

int i;

for (i = 0; i < len; i++){

if ((string[i] >= 'a' && string[i] <= 'z') || (string[i] >= 'A' && string[i] <= 'Z') )

l += 1;

else

if ((string[i] >= '0' && string[i] <= '9'))

d += 1;

else

s += 1 }

if (d > 0){

pthread\_mutex\_lock(&d.m[0]);

\*(d.digits) += d;

pthread\_mutex\_unlock(&d.m[0]) }

if (l > 0){ pthread\_mutex\_lock(&d.m[1]); \*(d.letters) += l; pthread\_mutex\_unlock(&d.m[1]); }

if (s > 0){

pthread\_mutex\_lock(&d.m[2]);

\*(d.specials) += s;

pthread\_mutex\_unlock(&d.m[2]);

}

return NULL;

}

void init(data \*arg, phtread\_mutex\_t \*mtxs, int cd, int cl, int cs, char\* string){

arg->m = &mtxs;

arg->digits = cd;

arg->letters = cl;

arg->specials = cs;

arg->string = string;

}

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

int digits = 0, letters = 0, specials = 0;

int i;

pthread\_t T[argc - 1];

phtread\_mutex\_t mtxs[3];

data d[argc - 1];

for (i = 0; i < argc - 1; i++){

pthread\_create(&T[i], NULL, f, (void\*)&d[i];)

}

for (i = 0; i < argc - 1; i++){

pthread\_mutex\_init(&mtxs[i], NULL);

}

for (i = 0; i < argc - 1; i++){

init(&d[i], mtxs[i], digits, letters, specials, argv[i + 1]);

}

for (i = 0; i < argc - 1; i++){

pthread\_mutex\_destroy(mtxs[i]);

}

for (i = 0; i < argc - 1; i++){ pthread\_join(&T[i], NULL) }

printf("Number of digits: %d \n", digits);

printf("Number of letters: %d \n", letters);

printf("Number of specials char: %d \n", specials);

return 0;

}

**12.Vector frecventa cifre cmd line argmts**

#include <pthread.h>

#include <string.h>

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

typedef struct{

pthread\_mutex\_t\* mtx;

int\* freq;

int info;

} data;

void\* f(void\* a) {

data d = \*((data\*)a);

int digit;

int aux = d.info;

while(aux > 0) {

digit = aux % 10;

pthread\_mutex\_lock(d.mtx);

d.freq[digit] += 1;

pthread\_mutex\_unlock(d.mtx);

aux = aux / 10;

}

return NULL;

}

void initData(data\* d, pthread\_mutex\_t\* given\_mtx, int\* given\_freq, int given\_info) {

d->mtx = given\_mtx;

d->freq = given\_freq;

d->info = given\_info;

}

int main(int argc, char\*\* argv) {

int N = argc - 1;

int i;

pthread\_t T[N];

pthread\_mutex\_t mtx;

int\* freq = (int\*)malloc(sizeof(int) \* 10);

for(i = 0; i < 10; i++)

freq[i] = 0;

data\* arg = (data\*)malloc(sizeof(data) \* N);

for(i = 0; i < N; i++)

initData(&arg[i], &mtx, freq, atoi(argv[i+1]));

pthread\_mutex\_init(&mtx, NULL)

for(i = 0; i < N; i++)

pthread\_create(&T[i], NULL, f, (void\*)&arg[i]);

for(i = 0; i < N; i++)

pthread\_join(T[i], NULL);

printf("Frequency array\n");

for(i = 0; i < 10; i++)

printf("[%d]: %d\n", i, freq[i]);

pthread\_mutex\_destroy(&mtx);

free(freq);

free(arg);

return 0;

}

**13. 2 threads -> even nr on even pos, odd nr on odd pos**

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <pthread.h>

#include <time.h>

typedef struct {

pthread\_mutex\_t \*m; pthread\_cond\_t \*c;

int \*array, \*index;

} data;

int N;

void \*f1(void \*arg){

data d = \*((data\*)arg);

int i;

pthread\_mutex\_lock(d.m);

if (\*(d.index) % 2 != 0 )

pthread\_cond\_wait(d.c, d.m);

while(\*(d.index) < N){

int nr;

nr = (random() % 51 ) \* 2;

d.array[\*(d.index)] = nr;

\*(d.index) += 1;

printf("T1: ");

for (i = 0; i < \*(d.index); i++){

printf("%d ", d.array[i]);

}

printf("\n");

pthread\_cond\_signal(d.c);

if (\*(d.index) % 2 != 0 )

pthread\_cond\_wait(d.c, d.m);

}

pthread\_cond\_signal(d.c);

pthread\_mutex\_unlock(d.m);

return NULL;

}

void \*f2(void \*arg){

data d = \*((data\*)arg);

int i;

pthread\_mutex\_lock(d.m);

if (\*(d.index) % 2 != 1 )

pthread\_cond\_wait(d.c, d.m);

while(\*(d.index) < N){

int nr;

nr = (random() % 51 );

d.array[\*(d.index)] = nr;

\*(d.index) += 1;

printf("T2: ");

for (i = 0; i < \*(d.index); i++){

printf("%d ", d.array[i]);

}

printf("\n");

pthread\_cond\_signal(d.c);

if (\*(d.index) % 2 != 1 )

pthread\_cond\_wait(d.c, d.m);

}

pthread\_cond\_signal(d.c);

pthread\_mutex\_unlock(d.m);

return NULL;

}

void init\_arg(data \*arg, pthread\_mutex\_t mutex, pthread\_cond\_t cond, int \*arr, int \*index) {

arg->m = &mutex;

arg->c = &cond;

arg->array = arr;

arg->index = index;

}

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

if (argc != 2){

printf("Provide just one argument! \n");

exit(1);

}

srandom(time(NULL));

N = atoi(argv[1]);

int \*array = malloc(sizeof(int) \* N);

int index = 0;

pthread\_mutex\_t m;

pthread\_cond\_t c;

pthread\_mutex\_init(&m, NULL);

pthread\_cond\_init(&c, NULL);

pthread\_t T[2];

data args[2];

init\_arg(&args[0], m, c, array, &index);

init\_arg(&args[1], m, c, array, &index);

pthread\_create(&T[0], NULL, f1, (void \*)&args[0]);

pthread\_create(&T[1], NULL, f2, (void \*)&args[1]);

pthread\_join(T[0], NULL);

pthread\_join(T[1], NULL);

pthread\_mutex\_destroy(&m);

pthread\_cond\_destroy(&c);

free(array);

printf("Done");

return 0;

}

**14.Upcase letters**

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void\* upcase(void\* arg) {

int i;

char\* string = \*((char\*\*)arg);

for (i = 0; i < strlen(string); i++){

if (string[i] >= 'a' && string[i] <= 'z'){

string[i] += 'A' - 'a';

}

}

printf("Thread finished: %s \n", string);

return NULL;

}

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

int i;

pthread\_t thrds[argc];

for (i = 0; i < argc; i++) {

if (0 > pthread\_create(&thrds[i], NULL, upcase, (void\*)&argv[i])){

perror("Error on create thread");

} }

for (i = 0; i < argc; i++) {

pthread\_join(thrds[i], NULL); }

printf("All threads finished \n");

return 0; }

**Model Horea 1**

#include <pthread.h>

#include <string.h>

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

typedef struct{ **//create struct**

int index;

int\* pos;

pthread\_mutex\_t m;

}data;

int \*array;

void\* f(void\* arg){

data d =\*((data\*)arg); **//convert data**

int nr1, nr2;

nr1 = random() % 999 + 1;

nr2 = random() % 999 + 1;

printf("T[%d]: with the numbers: %d and %d \n", d.index, nr1, nr2);

if (nr1 % 2 != nr2 % 2){

pthread\_mutex\_lock(&d.m); **//lock mutex**

array[\*(d.pos)] = nr1;

\*(d.pos) += 1; **//deferenciate pointers**

array[\*(d.pos)] = nr2;

\*(d.pos) += 1;

pthread\_mutex\_unlock(&d.m); } **//unlock mutex**

return NULL; }

void init\_data(data \*arg, int idx, int\* pos, pthread\_mutex\_t mtxs){

arg->m = mtxs;

arg->index = idx;

arg->pos = pos; }

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

if (argc != 2){

printf("Provide just one arg");

exit(1); }

srand(time(NULL));

int N = atoi(argv[1]);

int i;

int pos = 0;

array = malloc(sizeof(int) \* N \* 2);

data d[N];

pthread\_t T[N];

pthread\_mutex\_t m;

pthread\_mutex\_init(&m, NULL); **//Init Mutex**

for (i = 0; i < N; i++){

init\_data(&d[i], i, &pos, m); }

for (i = 0; i < N; i++){ **//init structs**

pthread\_create(&T[i], NULL, f, (void\*)&d[i]); } **//create thread**

for (i = 0; i < N; i++){

pthread\_join(T[i], NULL); } **//wait thread**

printf("Vector: ");

for (i = 0; i < pos; i++ ){

printf("%d ", array[i]); }

printf("\n");

pthread\_mutex\_destroy(&m); **//destroy thread**

free(array);

return 0; }

**Model Horea 2**

#include <pthread.h>

#include <string.h>

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

char\* array;

int contor;

typedef struct{

char\* arr;

pthread\_mutex\_t m;

}data;

void\* f(void\* arg){

int i;

data d = \*((data\*)arg);

int ok = 0;

for (i = 0; i < strlen(d.arr); i++){

if (d.arr[i] < '0' || d.arr[i] > '9')

ok = 1;

}

if (ok == 1){

pthread\_mutex\_lock(&d.m);

contor += 1;

pthread\_mutex\_unlock(&d.m);

}

else{

pthread\_mutex\_lock(&d.m);

strcat(array, d.arr);

pthread\_mutex\_unlock(&d.m);

}

return NULL; }

void init\_data(data \*arg, char\* string, pthread\_mutex\_t mtx){

arg->m = mtx;

arg-> arr = string; }

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

pthread\_t T[argc - 1];

pthread\_mutex\_t m;

contor = 0;

array = malloc(sizeof(char) \* (argc - 1) \* 10);

pthread\_mutex\_init(&m, NULL);

int i;

data d[argc -1];

for (i = 0; i < argc - 1; i++){

init\_data(&d[i], argv[i+1], m);

}

for (i = 0; i < argc - 1; i++){

if (0 > pthread\_create(&T[i], NULL, f, (void\*)&d[i])){

perror("Error on create thread");

}

}

for (i = 0; i < argc -1; i++){

pthread\_join(T[i], NULL);

}

printf("Contor: %d \n", contor);

printf("Vector: %s \n", array);

pthread\_mutex\_destroy(&m);

free(array);

return 0;

}

**Model Alina**

#include <pthread.h>

#include <string.h>

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

int c2;

int c3;

pthread\_mutex\_t mtx[2];

void\* f(void\* arg){

while (c2 < 5){

int nr;

nr = random() % 100 + 50;

printf("T[%d] generated: %d \n", \*(int\*)arg, nr);

if (nr < 100 ){

pthread\_mutex\_lock(&mtx[0]);

c2 += 1;

pthread\_mutex\_unlock(&mtx[0]);

}

else {

pthread\_mutex\_lock(&mtx[1]);

c3 += 1;

pthread\_mutex\_unlock(&mtx[1]);

}

}

return NULL;

}

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

pthread\_t T[10];

int i;

pthread\_mutex\_init(&mtx[0], NULL);

pthread\_mutex\_init(&mtx[1], NULL);

for (i = 0; i < 10; i++){

pthread\_create(&T[i], NULL, f, &i);

}

for (i = 0; i < 10; i++){

pthread\_join(T[i], NULL);

}

printf("counter for double digits: %d \n", c2);

printf("counter for triple digits: %d \n", c3);

pthread\_mutex\_destroy(&mtx[0]);

pthread\_mutex\_destroy(&mtx[1]);

return 0;

}

**Model 1**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

typedef struct {

int idx;

pthread\_mutex\_t \*m;

int \*p;

} data;

typedef struct {

int idx, nr\_1, nr\_2;

} info;

void\* f(void\* arg){

data d = \*((data\*)arg);

int nr\_1 = random() % 101;

int nr\_2 = random() % 101;

info x;

x.idx = d.idx;

x.nr\_1 = nr\_1;

x.nr\_2 = nr\_2;

printf("Thread %d has generated %d and %d\n", d.idx, nr\_1, nr\_2);

pthread\_mutex\_lock(d.m);

if(0 > write(d.p[1], &x, sizeof(info))) {

perror("Error sending data");

}

pthread\_mutex\_unlock(d.m);

return NULL;

}

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

if (argc != 2){

printf("Provide one argument \n");

exit(1);

}

int p[2];

int n = atoi(argv[1]);

if (0 > pipe(p)) {

perror("Error creating pipe: ");

exit(1);

}

if (0 == fork()){

close(p[1]);

int i;

for (i = 0; i < n; i++){

info x;

if(0 > read(p[0], &x, sizeof(info))) {

perror("Error reading data");

}

printf("Thread %d - average: %f\n", x.idx, (double)(x.nr\_1 + x.nr\_2) / 2);

}

close(p[0]);

exit(0);

}

else{

close(p[0]);

int i;

pthread\_t T[n];

data d[n];

pthread\_mutex\_t mutex;

pthread\_mutex\_init(&mutex, NULL);

for (i = 0; i < n; i++){

d[i].idx = i;

d[i].m = &mutex;

d[i].p = p;

pthread\_create(&T[i], NULL, f, (void\*)&d[i]);

//close(p[1]);

//wait(0);

}

for (i = 0; i < n; i++){

pthread\_join(T[i], NULL);

}

close(p[1]);

pthread\_mutex\_destroy(&mutex);

wait(0);

}

return 0;

}

**Model 2**

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <pthread.h>

int sum;

char\* vowels;

typedef struct {

pthread\_mutex\_t \*m;

char\* str;

int\* count;

} data;

void\* f1(void\* arg){

data d = \*((data\*)arg);

int i;

for (i = 0; i < strlen(d.str); i++){

if (d.str[i] == 'a' || d.str[i] == 'e' || d.str[i] == 'i' || d.str[i] == 'o' || d.str[i] == 'u'){

pthread\_mutex\_lock(d.m);

//printf("%s", vowels);

vowels[\*(d.count)] = d.str[i];

printf("%s", vowels);

\*(d.count) += 1;

pthread\_mutex\_unlock(d.m);

}

}

return NULL;

}

void\* f2(void\* arg){

data d = \*((data\*)arg);

int i;

for (i = 0; i < strlen(d.str); i++){

if (d.str[i] >= '0' && d.str[i] <= '9'){

pthread\_mutex\_lock(d.m);

sum += (d.str[i] - '0');

pthread\_mutex\_unlock(d.m);

}

}

return NULL;

}

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

if (argc == 1){

printf("Please provide arguments\n");

exit(1);

}

vowels = (char\*)malloc(50\*sizeof(char));

int p1[2], p2[2];

pipe(p1);

pipe(p2);

if (0 == fork()){

close(p2[0]); close(p2[1]);

close(p1[0]);

pthread\_mutex\_t m;

pthread\_t T1[argc -1];

data d[argc - 1];

pthread\_mutex\_init(&m, NULL);

int i;

for (i=0; i< argc-1; i++){

d[i].m = &m;

d[i].str = argv[i+1];

d[i].count = 0;

}

for (i = 0; i < argc -1; i++){

pthread\_create(&T1[i], NULL, f1, (void\*)&d[i]);

}

for (i = 0; i < argc-1; i++){

pthread\_join(T1[i], NULL);

}

int lun;

lun = strlen(vowels);

write(p1[1], &lun, sizeof(int));

write(p1[1], vowels, lun\*sizeof(char));

close(p1[1]);

}

else

if (0 == fork()){

close(p1[0]); close(p1[1]);

close(p2[0]);

pthread\_mutex\_t m;

pthread\_t T2[argc -1];

data d[argc - 1];

pthread\_mutex\_init(&m, NULL);

int i;

for (i=0; i< argc-1; i++){

d[i].m = &m;

d[i].str = argv[i+1];

d[i].count = 0;

}

for (i = 0; i < argc-1; i++){

pthread\_create(&T2[i], NULL, f2, (void\*)&d[i]);

}

for (i = 0; i < argc-1; i++){

pthread\_join(T2[i], NULL);

}

int aux;

aux = sum;

write(p2[1],&aux, sizeof(int));

close(p2[1]);

}

else{

int suma;

int dim;

close(p1[1]); close(p2[1]);

read(p1[0], &dim, sizeof(int));

char\* str = (char\*)malloc((dim+1)\*sizeof(char));

read(p1[0], &str, dim\*sizeof(char));

read(p2[0], &suma, sizeof(int));

wait(0);

wait(0);

printf("Vowesl are: %s \n", str);

printf("Sum of digits is %d \n", suma);

close(p1[0]); close(p2[0]);

}

return 0;

}