### ESCRITOR.C ###

#**include** "header.h"

#**include** "escritormon.h"

#**include** <string.h>

#**define** **NUM\_THREADS** 3

**int** lock, with\_errors, terminate = 1;

**void** **usr1\_handler**(){

if(lock)

lock=0;

else

lock=1;

printf("\n\nLock Status: %s\n\n", lock ?"locked":"unlocked");

}

**void** **usr2\_handler**(){

if(with\_errors)

with\_errors=0;

else

with\_errors=1;

printf("\n\nError Status: %s\n\n", with\_errors ?"With Errors":"Without Errors");

}

**void** **stop\_handler**(){

if(terminate)

terminate = 0;

printf("\n\nTerminate Status: %s\n\n", terminate ?"Program is running. . .":"Program will terminate soon. . .");

}

**void**\* **write\_file**(){

char filename[] = FILE\_NAME;

char letters[NUM\_STRINGS][NUM\_CHARS] = {AAA,BBB,CCC,DDD,EEE,FFF,GGG,HHH,III,JJJ};

int i,file,lock\_status,letter;

int save\_lock;

printf("Executing write thread\n");

while(terminate){

save\_lock = lock;

filename[X\_POS] = (char)(((int)'0')+ rand() % NUM\_FILES);

file = open(filename, O\_RDWR | O\_CREAT, S\_IRWXU | S\_IROTH);

if(save\_lock){

lock\_status = flock(file, LOCK\_EX);

if(lock\_status == -1){

perror("Error locking");

close(file);

exit(-1);

}

}else{

}

if(with\_errors){

for(i=0; i<NUM\_LINES;i++){

write(file,letters[**rand**()% NUM\_STRINGS], NUM\_CHARS-1);

}

}else{

letter = rand()% NUM\_STRINGS;

for(i=0; i<NUM\_LINES;i++){

write(file,letters[letter], NUM\_CHARS-1);

}

}

if(save\_lock)

flock(file,LOCK\_UN);

close(file);

}

pthread\_exit(NULL);

}

**int** **main**(){

struct sigaction usr1, usr2, stop;

struct timeval tvstart;

int i, j, status;

pthread\_t threads[NUM\_THREADS];

gettimeofday(&tvstart, NULL);

srand((tvstart.tv\_sec) \* 1000 + (tvstart.tv\_usec) / 1000);

usr1.sa\_handler = usr1\_handler;

sigemptyset(&usr1.sa\_mask);

usr1.sa\_flags = 0;

sigaddset(&usr1.sa\_mask,SIGUSR1);

usr2.sa\_handler = usr2\_handler;

sigemptyset(&usr2.sa\_mask);

usr2.sa\_flags = 0;

sigaddset(&usr2.sa\_mask,SIGUSR2);

stop.sa\_handler = stop\_handler;

sigemptyset(&stop.sa\_mask);

sigaddset(&stop.sa\_mask,SIGTSTP);

stop.sa\_flags = 0;

sigaction(SIGUSR1, &usr1, NULL);

sigaction(SIGUSR2, &usr2, NULL);

sigaction(SIGTSTP, &stop, NULL);

lock = 0;

with\_errors = 0;

for(i=0; i < NUM\_THREADS; i++){

status = pthread\_create(&threads[i],NULL, write\_file, NULL);

if(status != 0){

printf("Oops. pthread create returned error code %d\n",status);

exit(-1);

}

}

for(j = 0; j < NUM\_THREADS ; j++){

pthread\_join(threads[j],NULL);

printf("Thread [%d] returned with value.\n",j);

}

return 0;

}

### ESCRITOR.H ###

#**ifndef** \_\_ESCRITOR\_H\_\_

#define \_\_ESCRITOR\_H\_\_

#define NUM\_CYCLES 1024

int randomNumber(int max);

int choose\_lock\_file(char \*filename);

void close\_unlock\_file(int file);

int choose\_letter();

#**endif**

### LEITOR.C ###

#**include** "header.h"

#**include** <pthread.h>

#**include** <string.h>

#**include** <semaphore.h>

**void** **close\_unlock\_file**(file){

flock(file,LOCK\_UN);

close(file);

}

**int** **confirma\_string**(char \* buffer, char letters[NUM\_STRINGS][NUM\_CHARS]){

/\*Valida a primeira string do ficheiro\*/

int i;

for(i = 0; i < NUM\_STRINGS; i++){

if(i == NUM\_STRINGS)

return -1;

if(**strcmp**(letters[i],buffer)==0)

return 0;

}

return -1;

}

**int** **open\_lock\_file**(char \*filename){

/\*Abre o ficheiro pedido\*/

int lock\_status, file;

file = open(filename, O\_RDONLY);

/\*Verifica se o ficheiro esta a ser escrito\*/

lock\_status = flock(file, LOCK\_SH | LOCK\_NB);

if(lock\_status == -1){

if( errno == EWOULDBLOCK){

printf("Ficheiro esta a ser usado\n");

}

}

/\*Executa Shared Lock no ficheiro\*/

lock\_status = flock(file, LOCK\_SH);

if(lock\_status == -1){

perror("Error locking @open\_lock\_file");

close(file);

exit(-1);

}

return file;

}

**int** index = 0;

**char** shbuffer[BUFFER\_SIZE][FILE\_NAME\_SIZE];

**pthread\_mutex\_t** mutex;

sem\_t sem\_leitor;

**void**\* **read\_file**() {

int contador = 0;

int file, fline;

char buffer[NUM\_CHARS] = "";

char first[NUM\_CHARS] = "";

char filename[FILE\_NAME\_SIZE];

while(1) {

sem\_wait(&sem\_leitor);

pthread\_mutex\_lock(&mutex);

file = open\_lock\_file(shbuffer[--index]); /\*"consome" uma string do buffer partilhado\*/

strcpy(filename,shbuffer[index]); /\*Guarda o nome do ficheiro\*/

memset(shbuffer[index],0,FILE\_NAME\_SIZE); /\*Limpa o que acabou de retirar\*/

pthread\_mutex\_unlock(&mutex);

read(file, buffer, NUM\_CHARS-1);

strcpy(first,buffer);

if((**confirma\_string**(first, letters)) != 0){

close\_unlock\_file(file);

printf("Something went wrong (Confirma String @read\_file...\n");

pthread\_exit((void\*)-1);

}

contador = 0;

while(**strcmp**(buffer,first) == 0) {

fline = read(file,buffer,NUM\_CHARS-1);

if(**strcmp**(buffer,first) != 0) {

printf("Ficheiro %s incorrecto!\n",filename);

break;

}

contador++;

if(fline == -1) {

perror("");

printf("Error in read (contador em %d)\n",contador);

break;

}

if(fline == 0){

if (contador != NUM\_LINES) {

close\_unlock\_file(file);

printf("Ficheiro %s... Check!\n",filename);

break;

}

}

}

}

printf("Something went wrong (Durante a leitura do ficheiro)...\n");

close\_unlock\_file(file);

pthread\_exit((void\*)-1);

}

**int** **inputIsNotValid**(char input[]) {

char checkfile[FILE\_NAME\_SIZE];

strcpy(checkfile,input);

checkfile[X\_POS] = 'x';

if((**strcmp**(checkfile,FILE\_NAME) == 0) && input[X\_POS] >= '0' && input[X\_POS] < '0' + NUM\_FILES)

return 0;

return 1;

}

**int** **main**() {

struct timeval tvstart;

int retvalue;

int status, i, j, inp;

char input;

char buffer[FILE\_NAME\_SIZE];

pthread\_t threads[NUM\_THREADS\_LEITOR];

gettimeofday(&tvstart, NULL);

/\*Inicializa o semaforo (partilhado por threads)\*/

if (**sem\_init**(&sem\_leitor,THREAD\_SHARED,0) != 0) {

perror("\nSemaphore init failed\n");

exit(-1);

}

/\*Inicializa o mutex\*/

if (**pthread\_mutex\_init**(&mutex, NULL) != 0) {

perror("\nMutex init failed\n");

exit(-1);

}

for(i=0; i < NUM\_THREADS\_LEITOR; i++) {

printf("Main function here. Creating thread %d\n", i);

status = pthread\_create(&threads[i], NULL, read\_file, NULL);

if (status != 0) {

printf("Oops. pthread create returned error code %d\n", status);

exit(-1);

}

}

i = 0;

while(1) {

inp = read(STDIN\_FILENO, &input, sizeof(char));

if(i == FILE\_NAME\_SIZE-1) { /\*Quando chega a ultima posicao\*/

buffer[i] = '\0'; /\*"Fecha" a string\*/

if(**inputIsNotValid**(buffer)) {

memset(buffer,0,FILE\_NAME\_SIZE);

i = 0;

printf("Input was not valid!\n");

continue;

}

pthread\_mutex\_lock(&mutex);

strcpy(shbuffer[index],buffer); /\*Copia para o buffer partilhado\*/

index = (index + 1) % BUFFER\_SIZE;

sem\_post(&sem\_leitor); /\*Assinala o semaforo do leitor\*/

memset(buffer,0,FILE\_NAME\_SIZE); /\*Limpa o buffer\*/

i = 0; /\*Faz reset no buffer\*/

pthread\_mutex\_unlock(&mutex);

}

if(inp == END) {

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

printf("[%d] %s\n",i,shbuffer[i]);

i = 0;

break;

}

if(input == '\n' || input == ' ') {

i = 0;

}

else {

buffer[i++] = input; /\*Coloca o caracter lido no input\*/

}

}

for(j = 0; j < NUM\_THREADS\_LEITOR; j++) {

pthread\_join(threads[j],(void\*\*)&retvalue);

printf("Thread[%d] returned with value %d.\n",j,retvalue);

}

exit(0);

}

### MONITOR2.C ###

#include "header.h"

#include <signal.h>

#define INPUT fd[0]

#define OUTPUT fd[1]

**int** inputIsNotValid(char input[]) {

char checkfile[FILE\_NAME\_SIZE];

strcpy(checkfile,input);

checkfile[X\_POS] = 'x';

if((**strcmp**(checkfile,FILE\_NAME) == 0) && input[X\_POS] >= '0' && input[X\_POS] < '0' + NUM\_FILES)

return 0;

return 1;

}

**int** main(){

int fd[2];

int i, status, pid\_esc, pid\_leitor;

char buffer[FILE\_NAME\_SIZE];

char input;

char send[1];

if(**pipe**(fd) == -1) {

perror("pipe");

exit(EXIT\_FAILURE);

}

pid\_esc = fork();

/\* ######### ESCRITOR ########\*/

if(pid\_esc == 0){

execl("escritormon","escritormon",NULL);

}

else{

pid\_leitor = fork();

/\* ####### LEITOR #######\*/

if(pid\_leitor == 0) {

close(OUTPUT);

if(**dup2**(INPUT,0) == -1)

perror("dup2");

execl("leitormon","leitormon",NULL);

}

/\* ######## PAI ########\*/

else{

i = 0;

while(1) {

read(STDIN\_FILENO, &input, sizeof(char));

send[0] = input;

write(OUTPUT,send,sizeof(char));

if(input == '\n') {

buffer[i] = '\0';

if(**strcmp**(buffer,IL) == 0) {

kill(pid\_esc,SIGUSR1);

}

else if(**strcmp**(buffer,IE) == 0) {

kill(pid\_esc,SIGUSR2);

}

else if(**strcmp**(buffer,EXIT) == 0) {

printf("Exiting...\n");

kill(pid\_esc,SIGTSTP);

kill(pid\_leitor,SIGTSTP);

wait(&status);

return 0;

}

memset(buffer,0,FILE\_NAME\_SIZE);

i=0;

continue;

}

if(input == ' ') {

i = 0;

continue;

}

else {

buffer[i++] = input;

}

}

wait(&status);

}

}

return 0;

}

### HEADER.H ###

#ifndef \_\_HEADER\_H\_\_

#define \_\_HEADER\_H\_\_

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/file.h>

#include <sys/time.h>

Makefile

all: escritormon monitor leitormon

escritormon: escritormon.c escritormon.h header.h

gcc -g -Wall -pedantic -o escritormon -pthread escritormon.c

monitor: monitor2.c

gcc -Wall -pedantic -o monitor2 monitor2.c header.h

leitormon: leitormon.c

gcc -pedantic -ansi -Wall -o leitormon -pthread leitormon.c

#include <errno.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <pthread.h>

#include <signal.h>

#define NUM\_FILES 5

#define NUM\_STRINGS 10

#define NUM\_CHARS 11

#define NUM\_LINES 1024

#define X\_POS 7

#define EXIT "sair"

#define IL "il"

#define IE "ie"

#define AAA "aaaaaaaaa\n"

#define BBB "bbbbbbbbb\n"

#define CCC "ccccccccc\n"

#define DDD "ddddddddd\n"

#define EEE "eeeeeeeee\n"

#define FFF "fffffffff\n"

Grupo 064

Gonçalo Fialho nº 79112

Pedro Santos nº 78328

Gonçalo Ferreira nº 78596

#define GGG "ggggggggg\n"

#define HHH "hhhhhhhhh\n"

#define III "iiiiiiiii\n"

#define JJJ "jjjjjjjjj\n"

#define FILE\_NAME "SO2014-x.txt"

#define FILE\_NAME\_SIZE 13

#define NUM\_THREADS\_LEITOR 5

#define BUFFER\_SIZE 10

#define THREAD\_SHARED 0

#define END 0

int open\_file(char \*filename);

int randomNumber(int max);

int choose\_file(char \*filename);

int choose\_letter();

char letters[NUM\_STRINGS][NUM\_CHARS] = {AAA,BBB,CCC,DDD,EEE,FFF,GGG,HHH,III,JJJ};

#endif