```
### 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;
}
```

```c
### 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>
#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"
#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

```

```makefile
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
```

Grupo 064

Gonçalo Fialho nº 79112

Pedro Santos nº 78328

Gonçalo Ferreira nº 78596