

```

1  ### ESCRITOR.C ###
2  #include "header.h"
3  #include "escritormon.h"
4  #include <string.h>
5  #define NUM_THREADS 3
6  int lock, with_errors, terminate = 1;
7
8  void usr1_handler(){
9      if(lock)
10         lock=0;
11     else
12         lock=1;
13     printf("\n\nLock Status: %s\n\n", lock
14 ?"locked":"unlocked");
15 }
16
17 void usr2_handler(){
18     if(with_errors)
19         with_errors=0;
20     else
21         with_errors=1;
22     printf("\n\nError Status: %s\n\n", with_errors
23 ?"With Errors":"Without Errors");
24 }
25
26 void stop_handler(){
27     if(terminate)
28         terminate = 0;
29     printf("\n\nTerminate Status: %s\n\n", terminate
30 ?"Program is running. . .":"Program will terminate
31 soon. . .");
32 }
33
34 void* write_file(){
35     char filename[] = FILE_NAME;
36     char letters[NUM_STRINGS][NUM_CHARS] =
37     {AAA,BBB,CCC,DDD,EEE,FFF,GGG,HHH,III,JJJ};
38     int i,file,lock_status,letter;
39     int save_lock;
40     printf("Executing write thread\n");
41     while(terminate){
42         save_lock = lock;
43         filename[X_POS] = (char)((((int)'0')+ rand() %
44 NUM_FILES);
45         file = open(filename, O_RDWR | O_CREAT, S_IRWXU
46 | S_IROTH);
47
48         if(save_lock){
49             lock_status = flock(file, LOCK_EX);
50             if(lock_status == -1){
51                 perror("Error locking");
52                 close(file);
53                 exit(-1);
54             }
55         }
56         else{
57
58
59             if(with_errors){
60                 for(i=0; i<NUM_LINES;i++){
61                     write(file,letters[rand()%
62 NUM_STRINGS], NUM_CHARS-1);
63                 }
64             }
65             else{
66                 letter = rand()% NUM_STRINGS;
67                 for(i=0; i<NUM_LINES;i++){
68                     write(file,letters[letter], NUM_CHARS-
69 1);
70                 }
71             }
72
73             if(save_lock)
74                 flock(file, LOCK_UN);
75             close(file);
76         }
77         pthread_exit(NULL);
78     }
79
80 int main(){
81     struct sigaction usr1, usr2, stop;
82     struct timeval tvstart;
83     int i, j, status;
84     pthread_t threads[NUM_THREADS];
85     gettimeofday(&tvstart, NULL);
86     srand((tvstart.tv_sec) * 1000 + (tvstart.tv_usec) /
87 1000);
88
89     usr1.sa_handler = usr1_handler;
90     sigemptyset(&usr1.sa_mask);
91     usr1.sa_flags = 0;
92     sigaddset(&usr1.sa_mask,SIGUSR1);
93
94     usr2.sa_handler = usr2_handler;
95     sigemptyset(&usr2.sa_mask);
96     usr2.sa_flags = 0;
97     sigaddset(&usr2.sa_mask,SIGUSR2);
98
99
100    stop.sa_handler = stop_handler;
101    sigemptyset(&stop.sa_mask);
102    sigaddset(&stop.sa_mask,SIGTSTP);
103    stop.sa_flags = 0;
104
105
106    sigaction(SIGUSR1, &usr1, NULL);
107    sigaction(SIGUSR2, &usr2, NULL);
108    sigaction(SIGTSTP, &stop, NULL);
109
110    lock = 0;
111    with_errors = 0;
112
113    for(i=0; i < NUM_THREADS; i++){
114        status = pthread_create(&threads[i],NULL,
115 write_file, NULL);
116
117        if(status != 0){
118            printf("Oops. pthread create returned error
119 code %d\n",status);
120            exit(-1);
121        }
122    }
123
124    for(j = 0; j < NUM_THREADS ; j++){
125
126        pthread_join(threads[j],NULL);
127
128        printf("Thread [%d] returned with value.\n",j);
129    }
130
131    return 0;
132 }
133
134 ### ESCRITOR.H ###
135 #ifndef __ESCRITOR_H__
136 #define __ESCRITOR_H__
137 #define NUM_CYCLES 1024
138 int randomNumber(int max);
139 int choose_lock_file(char *filename);
140 void close_unlock_file(int file);
141 int choose_letter();
142 #endif

```

```

141 ### LEITOR.C ###
142 #include "header.h"
143 #include <pthread.h>
144 #include <string.h>
145 #include <semaphore.h>
146
147 void close_unlock_file(file){
148     flock(file,LOCK_UN);
149     close(file);
150 }
151
152
153 int confirma_string(char * buffer, char
154 letters[NUM_STRINGS][NUM_CHARS]){
155
156     /*Valida a primeira string do ficheiro*/
157
158     int i;
159     for(i = 0; i < NUM_STRINGS; i++){
160         if(i == NUM_STRINGS)
161             return -1;
162         if(strcmp(letters[i],buffer)==0)
163             return 0;
164     }
165     return -1;
166 }
167
168
169 int open_lock_file(char *filename){
170
171     /*Abre o ficheiro pedido*/
172
173     int lock_status, file;
174     file = open(filename, O_RDONLY);
175
176
177     /*Verifica se o ficheiro esta a ser escrito*/
178     lock_status = flock(file, LOCK_SH | LOCK_NB);
179     if(lock_status == -1){
180         if( errno == EWOULDBLOCK){
181             printf("Ficheiro esta a ser usado\n");
182         }
183     }
184
185     /*Executa Shared Lock no ficheiro*/
186
187     lock_status = flock(file, LOCK_SH);
188     if(lock_status == -1){
189         perror("Error locking @open_lock_file");
190         close(file);
191         exit(-1);
192     }
193     return file;
194 }
195
196 int index = 0;
197 char shbuffer[BUFFER_SIZE][FILE_NAME_SIZE];
198 pthread_mutex_t mutex;
199 sem_t sem_leitor;
200
201 void* read_file() {
202
203     int contador = 0;
204     int file, fline;
205     char buffer[NUM_CHARS] = "";
206     char first[NUM_CHARS] = "";
207     char filename[FILE_NAME_SIZE];
208
209     while(1) {
210         sem_wait(&sem_leitor);
211         pthread_mutex_lock(&mutex);
212
213         file = open_lock_file(shbuffer[--index]);
214         /*"consome" uma string do buffer partilhado*/
215         strcpy(filename,shbuffer[index]);
216         /*Guarda o nome do ficheiro*/
217         memset(shbuffer[index],0,FILE_NAME_SIZE);
218         /*Limpa o que acabou de retirar*/
219
220
221         pthread_mutex_unlock(&mutex);
222
223         read(file, buffer, NUM_CHARS-1);
224         strcpy(first,buffer);
225
226         if((confirma_string(first, letters)) != 0){
227             close_unlock_file(file);
228             printf("Something went wrong (Confirma
229 String @read_file...\n");
230             pthread_exit((void*)-1);
231         }
232
233         contador = 0;
234         while(strcmp(buffer,first) == 0) {
235             fline = read(file,buffer,NUM_CHARS-1);
236             if(strcmp(buffer,first) != 0) {
237                 printf("Ficheiro %s
238 incorrecto!\n",filename);
239                 break;
240             }
241             contador++;
242             if(fline == -1) {
243                 perror("");
244                 printf("Error in read (contador em
245 %d)\n",contador);
246                 break;
247             }
248             if(fline == 0){
249                 if (contador != NUM_LINES) {
250                     close_unlock_file(file);
251                     printf("Ficheiro %s...
252 Check!\n",filename);
253                     break;
254                 }
255             }
256         }
257     }
258
259     printf("Something went wrong (Durante a leitura do
260 ficheiro)...\n");
261     close_unlock_file(file);
262     pthread_exit((void*)-1);
263 }
264
265
266 int inputIsValid(char input[]) {
267     char checkfile[FILE_NAME_SIZE];
268     strcpy(checkfile,input);
269     checkfile[X_POS] = 'x';
270     if((strcmp(checkfile,FILE_NAME) == 0) &&
271 input[X_POS] >= '0' && input[X_POS] < '0' + NUM_FILES)
272         return 0;
273     return 1;
274 }
275
276
277
278 int main() {
279     struct timeval tvstart;
280     int retvalue;
281     int status, i, j, inp;
282     char input;
283     char buffer[FILE_NAME_SIZE];
284
285
286     pthread_t threads[NUM_THREADS_LEITOR];
287     gettimeofday(&tvstart, NULL);
288
289     /*Inicializa o semaforo (partilhado por threads)*/
290     if (sem_init(&sem_leitor,THREAD_SHARED,0) != 0) {
291         perror("\nSemaphore init failed\n");
292         exit(-1);
293     }
294     /*Inicializa o mutex*/
295     if (pthread_mutex_init(&mutex, NULL) != 0) {
296         perror("\nMutex init failed\n");
297         exit(-1);
298     }

```

```

299
300
301
302     for(i=0; i < NUM_THREADS_LEITOR; i++) {
303
304         printf("Main function here. Creating thread
305 %d\n", i);
306         status = pthread_create(&threads[i], NULL,
307 read_file, NULL);
308         if (status != 0) {
309             printf("Oops. pthread create returned error
310 code %d\n", status);
311             exit(-1);
312         }
313     }
314
315     i = 0;
316     while(1) {
317         inp = read(STDIN_FILENO, &input, sizeof(char));
318         if(i == FILE_NAME_SIZE-1) {
319             /*Quando chega a ultima posicao*/
320             buffer[i] = '\0';
321             /*"Fecha" a string*/
322             if(inputIsValid(buffer)) {
323                 memset(buffer,0,FILE_NAME_SIZE);
324                 i = 0;
325                 printf("Input was not valid!\n");
326                 continue;
327             }
328             pthread_mutex_lock(&mutex);
329             strcpy(shbuffer[index],buffer);    /*Copia
330 para o buffer partilhado*/
331             index = (index + 1) % BUFFER_SIZE;
332             sem_post(&sem_leitor);
333             /*Assinala o semaforo do leitor*/
334             memset(buffer,0,FILE_NAME_SIZE);    /*Limpa
335 o buffer*/
336             i = 0;                                /*Faz
337 reset no buffer*/
338             pthread_mutex_unlock(&mutex);
339         }
340         if(inp == END) {
341             for(i = 0; i < index; i++)
342                 printf("[%d] %s\n",i,shbuffer[i]);
343             i = 0;
344             break;
345         }
346         if(input == '\n' || input == ' ') {
347             i = 0;
348         }
349         else {
350             buffer[i++] = input;
351             /*Coloca o caracter lido no input*/
352         }
353     }
354
355     for(j = 0; j < NUM_THREADS_LEITOR; j++) {
356         pthread_join(threads[j],(void**)&retvalue);
357         printf("Thread[%d] returned with value
358 %d.\n",j,retvalue);
359     }
360     exit(0);
361 }
362
363
364 ### MONITOR2.C ###
365
366 #include "header.h"
367 #include <signal.h>
368
369 #define INPUT fd[0]
370 #define OUTPUT fd[1]
371
372 int inputIsValid(char input[]) {
373     char checkfile[FILE_NAME_SIZE];
374     strcpy(checkfile,input);
375     checkfile[X_POS] = 'x';
376     if((strcmp(checkfile,FILE_NAME) == 0) &&
377 input[X_POS] >= '0' && input[X_POS] < '0' + NUM_FILES)

```

```

378         return 0;
379     return 1;
380 }
381
382
383
384
385 int main(){
386     int fd[2];
387     int i, status, pid_esc, pid_leitor;
388     char buffer[FILE_NAME_SIZE];
389     char input;
390     char send[1];
391
392     if(pipe(fd) == -1) {
393         perror("pipe");
394         exit(EXIT_FAILURE);
395     }
396
397     pid_esc = fork();
398
399     /* ##### ESCRITOR #####/
400 if(pid_esc == 0){
401     execl("escritormon","escritormon",NULL);
402 }
403 else{
404     pid_leitor = fork();
405
406     /* ##### LEITOR #####/
407 if(pid_leitor == 0) {
408     close(OUTPUT);
409     if(dup2(INPUT,0) == -1)
410         perror("dup2");
411     execl("leitormon","leitormon",NULL);
412 }
413
414     /* ##### PAI #####/
415 else{
416     i = 0;
417     while(1) {
418         read(STDIN_FILENO, &input,
419 sizeof(char));
420         send[0] = input;
421         write(OUTPUT,send,sizeof(char));
422         if(input == '\n') {
423             buffer[i] = '\0';
424             if(strcmp(buffer,IL) == 0) {
425                 kill(pid_esc,SIGUSR1);
426             }
427             else if(strcmp(buffer,IE) == 0) {
428                 kill(pid_esc,SIGUSR2);
429             }
430             else if(strcmp(buffer,EXIT) == 0) {
431                 printf("Exiting...\n");
432                 kill(pid_esc,SIGTSTP);
433                 kill(pid_leitor,SIGTSTP);
434                 wait(&status);
435                 return 0;
436             }
437             memset(buffer,0,FILE_NAME_SIZE);
438             i=0;
439             continue;
440         }
441         if(input == ' ') {
442             i = 0;
443             continue;
444         }
445         else {
446             buffer[i++] = input;
447         }
448     }
449     wait(&status);
450 }
451 }
452 return 0;
453 }
454
455

```

```

456 ### HEADER.H ###
457 #ifndef __HEADER_H__
458 #define __HEADER_H__
459 #include <stdlib.h>
460 #include <string.h>
461 #include <stdio.h>
462 #include <fcntl.h>
463 #include <unistd.h>
464 #include <sys/types.h>
465 #include <sys/stat.h>
466 #include <sys/file.h>
467 #include <sys/time.h>
468 #include <errno.h>
469 #include <sys/types.h>
470 #include <sys/wait.h>
471 #include <pthread.h>
472 #include <signal.h>
473 #define NUM_FILES 5
474 #define NUM_STRINGS 10
475 #define NUM_CHARS 11
476 #define NUM_LINES 1024
477 #define X_POS 7
478 #define EXIT "sair"
479 #define IL "il"
480 #define IE "ie"
481 #define AAA "aaaaaaaa\n"
482 #define BBB "bbbbbbbb\n"
483 #define CCC "cccccccc\n"
484 #define DDD "dddddddd\n"
485 #define EEE "eeeeeeee\n"
486 #define FFF "ffffffff\n"
487 #define GGG "gggggggg\n"
488 #define HHH "hhhhhhh\n"
489 #define III "iiiiiii\n"
490 #define JJJ "jjjjjjj\n"
491 #define FILE_NAME "S02014-x.txt"
492 #define FILE_NAME_SIZE 13
493 #define NUM_THREADS_LEITOR 5
494 #define BUFFER_SIZE 10
495 #define THREAD_SHARED 0
496 #define END 0

```

```

497 int open_file(char *filename);
498 int randomNumber(int max);
499 int choose_file(char *filename);
500 int choose_letter();
501 char letters[NUM_STRINGS][NUM_CHARS] =
502 {AAA, BBB, CCC, DDD, EEE, FFF, GGG, HHH, III, JJJ};
503 #endif
504

```

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

Gonalo Fialho n 79112

Pedro Santos n 78328

Gonalo Ferreira n 78596