Makefile

```
all: escritor_par leitor_par monitor

escritor_par: escritor_par.c
	gcc -pthread -o escritor_par escritor_par.c
leitor_par: leitor_par.c
	gcc -pthread -o leitor_par leitor_par.c
monitor: monitor.c
	gcc -g -pthread -o monitor monitor.c
run: monitor
	./monitor
```

readwriter.h

```
#ifndef READWRITER_H
#define READWRITER_H

#define N_FILES			5
#define INDICE_ID_FILE		7
#define CYCLES			512
#define PERMISSION_CODE_R	 0444
#define PERMISSION_CODE_W	0644	/* Leitura e escrita para o
		utilizador (6), e apenas leitura para group e world (4) */
#define N_LINES			1024
#define N_THREAD_WRITE		2
#define N_THREAD_READ		8
#define N_BUFFER		10
#define N_INPUT			100

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <sys/file.h>
```



```
#include <sys/time.h>
#include <pthread.h>
#include <semaphore.h>
#include <signal.h>

#endif
```

monitor.c

```
#include "readwriter.h"

int main(){

	int pid_w, pid_r, len, i, finish = 0, status;
	int fds[2];
	char input_buffer[100], *send;

	send = (char *) malloc(sizeof(char) * N_INPUT);

	pid_w = fork();

	if(pid_w < 0){
		perror("Fork Write: Failed!");
		return -1;
	}

	else if(pid_w == 0){
		execl("escritor_par", "escritor_par", NULL);
		perror("Exec Write: Failed!");
		exit(-1);
	}

	if(pipe(fds)){
		perror("Create pipe: Failed!\n");
		return -1;
	}
```

```
pid_r = fork();

if(pid_r < 0){
	perror("Fork Read: Failed!");
	return -1;
}

else if(pid_r == 0){
	close(0);
	dup(fds[0]);
	close(fds[0]);
	close(fds[1]);
	execl("leitor_par", "leitor_par", NULL);
	perror("Exec Read: Failed!");
	exit(-1);
}

else{

	close(fds[0]);

	while(1){
		memset(input_buffer, 0, N_INPUT);

		if(finish) break;

		read(0, input_buffer, N_INPUT - 1);

		send = strtok(input_buffer, " \n");

		while(send != NULL){

			if(strncmp(send, "sair", 4) == 0){
				kill(pid_w, SIGTSTP);
				close(fds[1]);
				finish = 1;
				break;
			}

			else if(strncmp(send, "il", 2) == 0){
				puts("Matar 1");
				kill(pid_w, SIGUSR1);
			}

			else if(strncmp(send, "ie", 2) == 0){
				puts("Matar 2");
				kill(pid_w, SIGUSR2);
			}

			else if(write(fds[1], send, N_INPUT) < 0){
				perror("Error to send!\n");
			}

			send = strtok(NULL, " \n");
		}
	}

	for(i = 0; i < 2; i++){
		wait(&status);
	}
}
	return 0;
}
```

escritor_par.c

```
#include "readwriter.h"

int finish = 0, mutex_enable = 1, error_enable = 0;
char *cadeia[10] = {"aaaaaaaaa\n","bbbbbbbbb\n","ccccccccc
\n","ddddddddd\n","eeeeeeeee\n",
                    "fffffffff\n","ggggggggg\n","hhhhhhhhh
\n","iiiiiiiii\n","jjjjjjjjj\n"};
```

```
pthread_mutex_t vec_mutex[N_FILES];

void usr1_handler(){
	mutex_enable++;
	puts("Received mutex toogle!");
}

void usr2_handler(){
	error_enable++;
	puts("Received error toogle!");
}

void stop_handler(){
	finish = 1;
	puts("Writer received stop!");
}

int escritor(){

	int file, escolhida, id_file, k, local_enable;
	char filename[13];

	while(1){

		local_enable = mutex_enable;

		if(finish) return 0;

		strcpy(filename, "SO2014-0.txt");

		id_file = rand() % N_FILES;
		filename[INDICE_ID_FILE] += id_file;

		file = open(filename, O_RDWR | O_CREAT,
PERMISSION_CODE_W);

		if (file < 0){
```



```
			perror("Open: Failed\n");
			return -1;
		}

		if(local_enable % 2){
			if(flock(file, LOCK_EX) < 0){
				perror("Flock on Lock: Failed.\n");
				close(file);
				return -1;
			}
		}

		if(pthread_mutex_lock(&vec_mutex[id_file]) != 0){
			perror("Lock mutex: Failed");
			flock(file, LOCK_UN);
			close(file);
			return -1;
		}

		escolhida = rand() % 10; /*Linha de caracteres
aleatoria*/

		if(error_enable % 2) write(file, "zzzzzzzzz\n", 10);
		else write(file, cadeia[escolhida], 10);

		for(k = 1; k < N_LINES; k++){

			if(write(file, cadeia[escolhida], 10) != 10){ /*Se
nao escrever 10 caracteres da erro*/
				perror("Write line: Failed\n");
				pthread_mutex_unlock(&vec_mutex[id_file]);
				flock(file, LOCK_UN);
					close(file);
					return -1;
			}
		}
```

```
                if(pthread_mutex_unlock(&vec_mutex[id_file]) != 0){
                        perror("Unlock mutex: Failed");
                        flock(file, LOCK_UN);
                        close(file);
                        return -1;
                }

                if(local_enable % 2){
                        if(flock(file, LOCK_UN) < 0){
                                perror("Flock on Lock: Failed.\n");
                                close(file);
                                return -1;
                        }
                }

                close(file);
        }

        return 0;
}

int main(){

        int i, j;
        pthread_t thread_array[N_THREAD_WRITE];

        struct sigaction new_action1;
        struct sigaction new_action2;
        struct sigaction new_action3;

        new_action1.sa_handler = usr1_handler;
        sigemptyset (&new_action1.sa_mask);
        sigaddset(&new_action1.sa_mask, SIGUSR1);
        new_action1.sa_flags = 0;
        sigaction(SIGUSR1, &new_action1, NULL);

        new_action2.sa_handler = usr2_handler;
```



```
        sigemptyset (&new_action2.sa_mask);
        sigaddset(&new_action2.sa_mask, SIGUSR2);
        new_action2.sa_flags = 0;
        sigaction(SIGUSR2, &new_action2, NULL);

        new_action3.sa_handler = stop_handler;
        sigemptyset (&new_action3.sa_mask);
        sigaddset(&new_action3.sa_mask, SIGTSTP);
        new_action3.sa_flags = 0;
        sigaction(SIGTSTP, &new_action3, NULL);

        srand(time(NULL));

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

                if(pthread_mutex_init(&vec_mutex[i], NULL) != 0){
                        perror("Initialize mutex: Failed");
                        return -1;
                }
        }

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

                if(pthread_create(&thread_array[i], NULL, (void *)
escritor, NULL) != 0){
                        perror("Create thread: Failed");
                        return -1;
                }

        }

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

                if(pthread_join(thread_array[i], NULL) != 0){
                        perror("Join thread: Failed");
                        return -1;
                }
```

```
	}

	return 0;
}
```

leitor_par.c

```
#include "readwriter.h"

pthread_mutex_t mutex;
sem_t semaphore[2];
char buffer[N_BUFFER][13], temp[13];
int ptr_read = 0, finish = 0;

int escolher_ficheiro(){ /*Escolhe ficheiro entre 5, aleatoriamente*/

	int id_file = rand() % 5;

	return id_file;
}

int leitor(){

	int file, i, wrong, x;
	char linha[10], primeira[10], filename[13];

	while(1){

		wrong = 0;
		i = 0;

		if(finish) break;

		if(sem_wait(&semaphore[0]) != 0){
			perror("Wait on semaphore0: Failed");
			return -1;
		}
```



```
		}

		if(finish) break;

		if(pthread_mutex_lock(&mutex) != 0){
			perror("Lock mutex: Failed");
			return -1;
		}

		strcpy(filename, buffer[ptr_read]);

		if(strcmp(filename, "") == 0){
			pthread_mutex_unlock(&mutex);
			continue;
		}

		ptr_read = (ptr_read + 1) % N_BUFFER;

		if(pthread_mutex_unlock(&mutex) != 0){
			perror("Unlock mutex: Failed");
			return -1;
		}

		if(sem_post(&semaphore[1]) != 0){
			perror("Post semaphore1: Failed");
			return -1;
		}

		file = open(filename, O_RDONLY |
PERMISSION_CODE_R);

		if (file < 0){
			perror("Open: Failed.\n");
			continue;
		}

		if(flock(file, LOCK_SH) < 0){
```

```
                perror("Flock on Lock: Failed.\n”);
                close(file);
                return -1;
        }

        x = read(file, primeira, 10);
        printf("%d\n", x);

        if(x < 10){     /*Le a primeira linha para comparacao*/
                perror("First Read: Failed.\n");
                flock(file, LOCK_UN);
                close(file);
                continue;
        }

        while(read(file, linha, 10) != 0){ /*Enquanto houver
linhas no ficheiro*/

                if(strncmp(primeira, linha, 10) != 0){
                        perror("Linha nao corresponde: Erro.\n");
    /*Se linha diferente da primeira da erro*/
                        wrong = 1;
                }

                if(wrong) break;

                i++;
        }

        if(flock(file, LOCK_UN) < 0){
                perror("Flock on Unlock: Failed.\n");
                close(file);
                return -1;
        }
```



```
                close(file);

                if(wrong) continue;

                if(i != (N_LINES - 1)){ /*Se acabou o ficheiro, sucesso*/
                        perror("Não acabou o ficheiro: Erro.\n");
                        return -1;
                }

                printf("Read success on %s!\n", filename);
        }

        return 0;
}

int main(){

        int i, receive;
        pthread_t thread_array[N_THREAD_READ];
        int ptr_write = 0;

        srand(time(NULL));

        if(pthread_mutex_init(&mutex, NULL) != 0){
                perror("Initialize mutex: Failed");
                return -1;
        }

        for(i = 0; i < 2; i++){

                if(sem_init(&semaphore[i], 0, i * N_BUFFER) != 0){
                        perror("Initialize semaphore: Failed");
                        return -1;
                }
        }

        for(i = 0; i < N_THREAD_READ; i++){
```

```
                if(pthread_create(&thread_array[i], NULL, (void *) leitor,
NULL) != 0){
                        perror("Create thread: Failed");
                        return -1;
                }
        }

        while(1){

                if(sem_wait(&semaphore[1]) != 0){
                        perror("Wait semaphore1: Failed");
                        return -1;
                }

                receive = read(0, temp, N_INPUT);

                if(receive < 0){
                        perror("Read stream: Failed!\n");
                        continue;
                }

                else if(receive == 0){
                        finish = 1;
                        for(i = 0; i < N_THREAD_READ; i++)
                                sem_post(&semaphore[0]);
                        break;
                }

                strcpy(buffer[ptr_write], temp);

                ptr_write = (ptr_write + 1) % N_BUFFER;

                if(sem_post(&semaphore[0]) != 0){
                        perror("Post semaphore0: Failed");
                        return -1;
                }
        }
```



```
        puts("Reader finished");

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

                if(pthread_join(thread_array[i], NULL) != 0){
                        perror("Join thread: Failed");
                }
        }

  return 0;
}
```