

# Sistemas Operativos

## Concurso 3 – Sinais/Threads/Mutex

67990 – José Diogo Ferrás

### Ex. I

a)

Código:

```
1#include <stdio.h>
2#include <signal.h>
3#include <unistd.h>
4#include <stdlib.h>
5#include <stdio.h>
6#include <signal.h>
7#include <unistd.h>
8#include <stdlib.h>
9
10int i = 0;
11
12void sighandler() {
13    i++;
14    if(i < 5){
15        printf("\nnice try\n");
16    }
17    else{
18        printf("\nyou got me this time\n");
19    }
20}
21
22int main (int argc, char* argv[]){
23    while(1){
24        if (signal(SIGINT,&sighandler) == SIG_ERR ){
25            printf ( "NO signal\n" );
26        }
27        if(i == 5){
28            break;
29        }
30    }
31    return 0 ;
32}
```

I/O:

```
zediogo@ubuntu:~/Documents/c3$ ./a.out
^C
nice try
^C
nice try
^C
nice try
^C
nice try
^C
you got me this time
zediogo@ubuntu:~/Documents/c3$
```

b)

Código:

```
1#include <stdlib.h>
2#include <stdio.h>
3#include <pthread.h>
4#include <unistd.h>
5
6int i = 0;
7char msg[] = "mensagem da thread ";
8
9void *routine(){
10    for(int j = 0; j < 19; j++){
11        printf("%c", msg[j]);
12        fflush(stdout);
13        sleep(1);
14    }
15    printf("%d\n", i);
16    sleep(1);
17}
18
19int main (int argc, char* argv[]){
20    pthread_t th[5];
21    for(i = 0; i < 5; i++){
22        if(pthread_create(th + i, NULL, &routine, NULL)){
23            return 1;
24        }
25        if (pthread_join(th[i], NULL)){
26            return 2;
27        }
28    }
29    return 0 ;
30}
```

I/O:

```
zedlogo@ubuntu:~/Documents/cs$ ./a.out
mensagem da thread 0
mensagem da thread 1
mensagem da thread 2
mensagem da thread 3
mensagem da thread 4
```

## Ex. II

Código:

```
1#include <stdlib.h>
2#include <stdio.h>
3#include <pthread.h>
4#include <unistd.h>
5#include <stdbool.h>
6
7bool end = false;
8char msg[30];
9
10void *routine0(){
11    printf("Thread 1: ");
12    fflush(stdout);
13    fgets(msg, sizeof(msg), stdin);
14}
15
16void *routine1(){
17    if(msg[0] == 'E' && msg[1] == '\n'){
18        end = true;
19    }
20    for(int i = 0; i < 30; i++){
21        if(msg[i] == '\n'){
22            break;
23        }
24        printf("Thread 2: ");
25        printf("%c\n", msg[i]);
26    }
27}
```

```

29 int main (int argc, char* argv[]){
30     pthread_t th0, th1;
31     while(true){
32         if(pthread_create(&th0, NULL, &routine0, NULL)){
33             return 1;
34         }
35         if (pthread_join(th0, NULL)){
36             return 2;
37         }
38         if(pthread_create(&th1, NULL, &routine1, NULL)){
39             return 1;
40         }
41         if (pthread_join(th1, NULL)){
42             return 2;
43         }
44         if(end){
45             break;
46         }
47     }
48     printf("Prog finished\n");
49     return 0;
50 }

```

I/O:

```

zediogo@ubuntu:~/Documents/c3$ ./a.out
Thread 1: abc
Thread 2: a
Thread 2: b
Thread 2: c
Thread 1: giro
Thread 2: g
Thread 2: i
Thread 2: r
Thread 2: o
Thread 1: EEE
Thread 2: E
Thread 2: E
Thread 2: E
Thread 1: E
Thread 2: E
Prog finished

```

### Ex. III

a)

```
1#include <pthread.h>
2#include <stdio.h>
3#include <stdlib.h>
4#include <unistd.h>
5#include <string.h>
6
7pthread_t tid[2];
8int counter;
9
10void * trythis(void* arg){
11    unsigned long i = 0;
12    counter += 1;
13    printf("\nJob %d has started\n", counter);
14    for(i = 0; i < (0xFFFFFFFF); i++);
15    printf("\nJob %d has finished\n", counter);
16    return NULL;
17}
18
19int main(void){
20    int i = 0;
21    int error;
22    while(i < 2){
23        error = pthread_create(&tid[i], NULL, &trythis, NULL);
24        if(error != 0){
25            printf("\nThread can't be created:[%s]", strerror(error));
26        }
27        i++;
28    }
29    pthread_join(tid[0], NULL);
30    pthread_join(tid[1], NULL);
31    return 0;
32}
```

```
zediogo@ubuntu:~/Documents/c3$ ./a.out
Job 1 has started
Job 2 has started
Job 2 has finished
Job 2 has finished
```

b)

Isto acontece pois o counter é incrementado pela primeira vez quando a primeira thread é criada, e é incrementado pela segunda vez quando a segunda thread é incrementada.

Como podemos observar, a segunda thread começa a executar a sua operação antes da primeira thread acabar e como counter é uma variável global, esta é alterada na thread 1 quando é alterada na thread 2, logo quando acaba de executar a tarefa da thread 1, o valor do counter já está a 2 e printa "Job 2 has finished" também.

c)

```
1#include <pthread.h>
2#include <stdio.h>
3#include <stdlib.h>
4#include <unistd.h>
5#include <string.h>
6
7pthread_t tid[2];
8int counter;
9pthread_mutex_t mutex;
10
11void * trythis(void* arg){
12    unsigned long i = 0;
13    pthread_mutex_lock(&mutex);
14    counter += 1;
15    printf("\nJob %d has started\n", counter);
16    for(i = 0; i < (0xFFFFFFFF); i++);
17    pthread_mutex_unlock(&mutex);
18    printf("\n Job %d has finished\n", counter);
19    return NULL;
20}
21
22int main(void){
23    pthread_mutex_init(&mutex, NULL);
24    int i = 0;
25    int error;
26    while(i < 2){
27        error = pthread_create(&tid[i], NULL, &trythis, NULL);
28        if(error != 0){
29            printf("\nThread can't be created:[%s]", strerror(error));
30        }
31        i++;
32    }
33    pthread_join(tid[0], NULL);
34    pthread_join(tid[1], NULL);
35    pthread_mutex_destroy(&mutex);
36    return 0;
37}
```