

Prática 2 - Sistemas Operacionais

Elton Mauricio Da Silva - RA 11201810955

link de arquivos: <https://github.com/eltonmds/ufabc_operational_systems>

1 Questao 1

Cenário de race condition

Para identificação do cenário, foi desenvolvido o código abaixo que cria duas threads que iteram 10 vezes chamando as funções 'aloca_recurso' e 'desaloca_recurso'.

```

1  #include <stdio.h>
2  #include <unistd.h>
3  #include <pthread.h>
4  #include <stdlib.h>
5  #include <sys/time.h>
6
7  #define MAX 10;
8  int RECURSO_MAXIMO = MAX;
9
10 // Cada thread utiliza as mesmas funções
11
12 // Essa função aloca a quantidade de recursos desejada, se existir
13 int aloca_recurso (int recurso_desejado) {
14     if (RECURSO_MAXIMO - recurso_desejado < 0)
15         return 0;
16     else {
17         RECURSO_MAXIMO -= recurso_desejado;
18         return 0;
19     }
20 }
21
22 // Essa função devolve uma certa quantidade de recursos alocados
23 int desaloca_recurso(int recurso_devolvido) {
24     if (RECURSO_MAXIMO + recurso_devolvido > 10)
25         return 0;
26
27     RECURSO_MAXIMO += recurso_devolvido;
28     return 0;
29 }
30
31
32 void *thread1() {
33     printf("Olá, sou a thread 1!\n");
34     int i = 0;
35     // Começo da sessão crítica
36     while (i < 10) {
37         aloca_recurso(i);
38         sleep((rand() % 10 + 1)/10);

```

```
39     desaloca_recurso(i);
40     printf("t1: RECURSO_MAXIMO: %d\n", RECURSO_MAXIMO);
41     i++;
42 }
43 // Fim da sess o cr tica
44 pthread_exit(NULL);
45 }
46
47 void *thread2() {
48     printf("O1 , sou a thread 2!\n");
49     int i = 0;
50     // Come o da sess o cr tica
51     while (i < 10) {
52         aloca_recurso(i);
53         sleep((rand() % 10+1)/10);
54         desaloca_recurso(i);
55         printf("t2: RECURSO_MAXIMO: %d\n", RECURSO_MAXIMO);
56     }
57     // Fim da sess o cr tica
58     pthread_exit(NULL);
59 }
60
61 int main() {
62     pthread_t thr[2];
63
64     int i = 0;
65     if(pthread_create(&thr[i], NULL, thread1, NULL)) {
66         printf("Ops... Houve um erro na cria o da thread %d.\n", i);
67         return 0;
68     }
69     i++;
70
71     if(pthread_create(&thr[i], NULL, thread2, NULL)) {
72         printf("Ops... Houve um erro na cria o da thread %d.\n", i);
73         return 0;
74     }
75     printf("Final: RECURSO_MAXIMO: %d\n", RECURSO_MAXIMO);
76 }
```

Explicação

Durante cada iteração, é printado o valor da variável `RECURSO_MAXIMO` e a thread em questão. Verificou-se que cada vez o programa retornava valores diferentes em ordens diferentes para a variável `RECURSO_MAXIMO`. Portanto, ocorre o race condition durante a execução do código. O race condition ocorre a partir do momento em que as threads entram em suas respectivas zonas críticas (explicitadas no código), onde, ao entrar nos respectivos loops, ocorro a concorrência entre elas.

Solução Proposta

A solução proposta utiliza o protocolo test-and-set.

Código

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <pthread.h>
4 #include <stdlib.h>
5 #include <sys/time.h>
6
7 #define MAX 10;
8 int RECURSO_MAXIMO = MAX;
9
10 typedef struct __lock_t {int flag;} lock_t;
11
12 void init(lock_t *mutex) {
13     mutex->flag = 0;
14 }
15
16 void lock(lock_t *mutex) {
17     while (mutex->flag == 1); //spin-wait
18     mutex->flag = 1;
19 }
20
21 void unlock(lock_t *mutex) {
22     mutex->flag = 0;
23 }
24
25 // Cada thread utiliza as mesmas funcoes
26
27 // Essa funcao aloca a quantidade de recursos desejada, se existir
28 int aloca_recurso (int recurso_desejado) {
29     if (RECURSO_MAXIMO - recurso_desejado < 0)
30         return 0;
31     else {
32         RECURSO_MAXIMO -= recurso_desejado;
33         return 0;
34     }
35 }
36
37 // Essa funcao devolve uma certa quantidade de recursos alocados
38 int desaloca_recurso(int recurso_devolvido) {
39     if (RECURSO_MAXIMO + recurso_devolvido > 10)
40         return 0;
41
42     RECURSO_MAXIMO += recurso_devolvido;
43     return 0;
```

```
44 }
45
46
47 void *thread1() {
48     printf("Ol , sou a thread 1!\n");
49     int i = 0;
50     // Come o da sess o cr tica
51     lock_t *MUTEX;
52     init(MUTEX);
53     lock(MUTEX);
54     while (i < 10) {
55         aloca_recurso(i);
56         sleep((rand() % 10 + 1)/10);
57         desaloca_recurso(i);
58         printf("t1: RECURSO_MAXIMO: %d\n", RECURSO_MAXIMO);
59         i++;
60     }
61     unlock(MUTEX);
62     // Fim da sess o cr tica
63     pthread_exit(NULL);
64 }
65
66 void *thread2() {
67     printf("Ol , sou a thread 2!\n");
68     int i = 0;
69     // Come o da sess o cr tica
70     lock_t *MUTEX;
71     init(MUTEX);
72     lock(MUTEX);
73     while (i < 10) {
74         aloca_recurso(i);
75         sleep((rand() % 10+1)/10);
76         desaloca_recurso(i);
77         printf("t2: RECURSO_MAXIMO: %d\n", RECURSO_MAXIMO);
78     }
79     unlock(MUTEX);
80     // Fim da sess o cr tica
81     pthread_exit(NULL);
82 }
83
84 int main() {
85     pthread_t thr[2];
86
87     int i = 0;
88     if(pthread_create(&thr[i], NULL, thread1, NULL)) {
89         printf("Ops... Houve um erro na cria o da thread %d.\n", i);
90         return 0;
```

```
91     }
92     i++;
93
94     if(pthread_create(&thr[i], NULL, thread2, NULL)) {
95         printf("Ops... Houve um erro na cria o da thread %d.\n", i);
96         return 0;
97     }
98     printf("Final: RECURSO_MAXIMO: %d\n", RECURSO_MAXIMO);
99 }
```