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'.

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

#define MAX 10;
int RECURSO_MAXIMO = MAX;

// Cada thread utiliza as mesmas fun   es

// Essa fun   o aloca a quantidade de recursos desejada, se existir
int aloca_recurso (int recurso_desejado) {
    if (RECURSO_MAXIMO - recurso_desejado < 0)
        return 0;
    else {
        RECURSO_MAXIMO -= recurso_desejado;
        return 0;
    }
}

// Essa fun   o devolve uma certa quantidade de recursos alocados
int desaloca_recurso(int recurso_devolvido) {
    if (RECURSO_MAXIMO + recurso_devolvido > 10)
        return 0;

    RECURSO_MAXIMO += recurso_devolvido;
    return 0;
}


void *thread1() {
    printf("Ol  , sou a thread 1!\n");
    int i = 0;
    // Come o da sess o cr tica
    while (i < 10) {
    aloca_recurso(i);
    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("Ol , 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 respectivios loops, ocorro a concorrência entre elas.

Soluçã Proposta

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

Código

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

#define MAX 10;
int RECURSO_MAXIMO = MAX;

typedef struct __lock_t {int flag;} lock_t;

void init(lock_t *mutex) {
    mutex->flag = 0;
}

void lock(lock_t *mutex) {
    while (mutex->flag == 1); //spin-wait
    mutex->flag = 1;
}

void unlock(lock_t *mutex) {
    mutex->flag = 0;
}

// Cada thread utiliza as mesmas fun   es

// Essa fun   o aloca a quantidade de recursos desejada, se existir
int aloca_recurso (int recurso_desejado) {
    if (RECURSO_MAXIMO - recurso_desejado < 0)
        return 0;
    else {
        RECURSO_MAXIMO -= recurso_desejado;
        return 0;
    }
}

// Essa fun   o devolve uma certa quantidade de recursos alocados
int desaloca_recurso(int recurso_devolvido) {
    if (RECURSO_MAXIMO + recurso_devolvido > 10)
        return 0;

    RECURSO_MAXIMO += recurso_devolvido;
    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  }
```