Problemas de concorrência na programação *multi-thread*

Alan Godoy Souza Mello
godoy@dca.fee.unicamp.br

23 de março de 2010

Atomicidade das operações

Thread i

```
1 (...)
2 var = i;
3 (...)
4 Retorna (var + 5 == var + 5);
```

- Há alguma chance deste código retornar *falso*?

- Suponha o seguinte código, para manter o controle da quantidade de *threads* interessadas em entrar em uma região crítica:

interessados = 0;

Thread i

```
1 enquanto verdade faça
2     interessados++;
3     entra_rc();
4     /* Região crítica */
5     sai_rc();
6     interessados--;
```

- Qual os valores máximo e mínimo que a variável **interessados** pode atingir?

Algoritmo do desempate

Thread 0

```
1 enquanto verdade faça
2     interesse[0] = 1;
3     ultimo = 0;
4     enquanto interesse[1] e ultimo
      == 0 faça
5     /* Região crítica */
6     interesse[0] = 0;
```

Thread 1

```
1 enquanto verdade faça
2     interesse[1] = 1;
3     ultimo = 1;
4     enquanto interesse[0] e ultimo
      == 1 faça
5     /* Região crítica */
6     interesse[1] = 0;
```

- Quais diferenças de comportamento ocorrem ao trocar as linhas **2** e **3**?

Thread 0

```
1  enquanto verdade faça
2      ultimo = 0;
3      interesse[0] = 1;
4      enquanto interesse[1] e ultimo
       == 0 faça
5      /* Região crítica */
6      interesse[0] = 0;
```

Thread 1

```
1  enquanto verdade faça
2      ultimo = 1;
3      interesse[1] = 1;
4      enquanto interesse[0] e ultimo
       == 1 faça
5      /* Região crítica */
6      interesse[1] = 0;
```

Usando o futex

lock = 0;

Thread 0

```
1 enquanto verdade faça
2     interesse[0] = 1;
3     ultimo = 0;
4     se interesse[1] e ultimo == 0
      então
5         futex_wait(&lock, 0);
6     /* Região crítica */
7     interesse[0] = 0;
8     futex_wake(&lock, 1);
```

Thread 1

```
1 enquanto verdade faça
2     interesse[1] = 1;
3     ultimo = 1;
4     se interesse[0] e ultimo == 1
      então
5         futex_wait(&lock, 0);
6     /* Região crítica */
7     interesse[1] = 0;
8     futex_wake(&lock, 1);
```

Thread 0

```
1 enquanto verdade faça
2     interesse[0] = 1;
3     ultimo = 0;
4     se interesse[1] e ultimo == 0
      então
5         futex_wait(&interesse[1], 1);
6     /* Região crítica */
7     interesse[0] = 0;
8     futex_wake(&interesse[0], 1);
```

Thread 1

```
1 enquanto verdade faça
2     interesse[1] = 1;
3     ultimo = 1;
4     se interesse[0] e ultimo == 1
      então
5         futex_wait(&interesse[0], 1);
6     /* Região crítica */
7     interesse[1] = 0;
8     futex_wake(&interesse[1], 1);
```

Thread 0

```
1 enquanto verdade faça
2     interesse[0] = 1;
3     ultimo = 0;
4     se interesse[1] e ultimo == 0
      então
5         futex_wait(&ultimo, 0);
6     /* Região crítica */
7     interesse[0] = 0;
8     futex_wake(&ultimo, 1);
```

Thread 1

```
1 enquanto verdade faça
2     interesse[1] = 1;
3     ultimo = 1;
4     se interesse[0] e ultimo == 1
      então
5         futex_wait(&ultimo, 1);
6     /* Região crítica */
7     interesse[1] = 0;
8     futex_wake(&ultimo, 1);
```

Thread 0

```
1 enquanto verdade faça
2     interesse[0] = 1;
3     ultimo = 0;
4     se interesse[1] e ultimo == 0
      então
5         futex_wait(&ultimo, 0);
6     /* Região crítica */
7     interesse[0] = 0;
8     ultimo = -1;
9     futex_wake(&ultimo, 1);
```

Thread 1

```
1 enquanto verdade faça
2     interesse[1] = 1;
3     ultimo = 1;
4     se interesse[0] e ultimo == 1
      então
5         futex_wait(&ultimo, 1);
6     /* Região crítica */
7     interesse[1] = 0;
8     ultimo = -1;
9     futex_wake(&ultimo, 1);
```

Thread 0 - Solução

```
1  enquanto verdade faça
2      interesse[0] = 1;
3      ultimo = 0;
4      futex_wake(&ultimo, 1);
5      enquanto interesse[1] e ultimo
       == 0 faça
6          futex_wait(&ultimo, 0);
7      /* Região crítica */
8      interesse[0] = 0;
9      ultimo = -1;
10     futex_wake(&ultimo, 1);
```

Thread 0 - Solução

```
1  enquanto verdade faça
2      interesse[1] = 1;
3      ultimo = 1;
4      futex_wake(&ultimo, 1);
5      enquanto interesse[0] e ultimo
       == 1 faça
6          futex_wait(&ultimo, 1);
7      /* Região crítica */
8      interesse[1] = 0;
9      ultimo = -1;
10     futex_wake(&ultimo, 1);
```

Thread 0

```
1  enquanto verdade faça
2      interesse[0] = 1;
3      ultimo[0] = 1;
4      ultimo[1] = 0;
5      cond = interesse[1] + ultimo[0];
6      futex_wake(&cond, 1);
7      futex_wait(&cond, 1 + 1);
8      /* Região crítica */
9      interesse[0] = 0;
10     futex_wake(&cond, 1);
```

Thread 1

```
1  enquanto verdade faça
2      interesse[1] = 1;
3      ultimo[1] = 1;
4      ultimo[0] = 0;
5      cond = interesse[0] + ultimo[1];
6      futex_wake(&cond, 1);
7      futex_wait(&cond, 1 + 1);
8      /* Região crítica */
9      interesse[1] = 0;
10     futex_wake(&cond, 1);
```

Thread 0

```
1 enquanto verdade faça
2     interesse[0] = 1;
3     ultimo = 0 + interesse[1];
4     futex_wake(&ultimo, 1);
5     futex_wait(&ultimo, 0 + 1);
6     /* Região crítica */
7     interesse[0] = 0;
8     futex_wake(&ultimo, 1);
```

Thread 1

```
1 enquanto verdade faça
2     interesse[1] = 1;
3     ultimo = 1 + interesse[0];
4     futex_wake(&ultimo, 1);
5     futex_wait(&ultimo, 1 + 1);
6     /* Região crítica */
7     interesse[1] = 0;
8     futex_wake(&ultimo, 1);
```

```
union {
    int cond;
    struct { uint16 ultimo; uint8 interesse[2]; } v;
} ctrl;
```

Thread 0

```
1  enquanto verdade faça
2      ctrl.v.interesse[0] = 1;
3      ctrl.v.ultimo = 0;
4      futex_wake(&ctrl.cond, 1);
5      aux[0].cond = ctrl.cond;
6      se ctrl.v.interesse[1] e
       ctrl.v.ultimo == 0 então
7          futex_wait(&ctrl.cond,
           aux[0].cond);
8      /* Região crítica */
9      ctrl.v.interesse[0] = 0;
10     futex_wake(&ctrl.cond, 1);
```

Thread 1

```
1  enquanto verdade faça
2      ctrl.v.interesse[1] = 1;
3      ctrl.v.ultimo = 1;
4      futex_wake(&ctrl.cond, 1);
5      aux[1].cond = ctrl.cond;
6      se ctrl.v.interesse[0] e
       ctrl.v.ultimo == 1 então
7          futex_wait(&ctrl.cond,
           aux[1].cond);
8      /* Região crítica */
9      ctrl.v.interesse[1] = 0;
10     futex_wake(&ctrl.cond, 1);
```

```
union {
    int cond;
    struct { uint16 ultimo; uint8 interesse[2]; } v;
} ctrl;
```

Thread 0 - Solução

```
1  enquanto verdade faça
2      ctrl.v.interesse[0] = 1;
3      ctrl.v.ultimo = 0;
4      futex_wake(&ctrl.cond, 1);
5      aux[0].cond = ctrl.cond;
6      enquanto ctrl.v.interesse[1] e
       ctrl.v.ultimo == 0 faça
7          futex_wait(&ctrl.cond,
           aux[0].cond);
8      /* Região crítica */
9      ctrl.v.interesse[0] = 0;
10     futex_wake(&ctrl.cond, 1);
```

Thread 1 - Solução

```
1  enquanto verdade faça
2      ctrl.v.interesse[1] = 1;
3      ctrl.v.ultimo = 1;
4      futex_wake(&ctrl.cond, 1);
5      aux[1].cond = ctrl.cond;
6      enquanto ctrl.v.interesse[0] e
       ctrl.v.ultimo == 1 faça
7          futex_wait(&ctrl.cond,
           aux[1].cond);
8      /* Região crítica */
9      ctrl.v.interesse[1] = 0;
10     futex_wake(&ctrl.cond, 1);
```