Programação Concorrente – Aula 7

Gilmar Gimenes Rodrigues

1 Fetch and Add

fetch_and_add(var, incr) por definição `<temp = var; var = var + inc; return temp;>`

No x86 temos add indivisível:

```
LOCK        faz com que a memória não seja cedida para outra CPU
ADD var, cons
```

Isto não é um Fetch and Add!

```
MOV reg, var
LOCK
ADD var, const
```

```
entra_regiao_critica() {
    int minha_vez;
    minha_vez = fetch_and_add(numero, 1);  /* entra_RC_test_and_set(); *
                                            * minha_vez = numero++;    *
                                            * sai_RC_test_and_set();   */
    while (proximo != minha_vez)
      ;
}

sai_regiao_critica() {
    proximo++;
}
```

1.1 Bakery Algorithm

Cada thread que espera tem um número, e ao chegar, uma thread escolhe um número estritamente maior que o das outras threads.

variável compartilhada:

`int vez[N];`

Zero em vez[i] indica que a thread i não está esperando.

Versão 0

```
vez[i] = maximo(vez) + 1; // i quer entrar
while (vez[i] > minimo_nao_nulo(vez))
  ;
regiao_critica();
vez[i] = 0;
regiao_nao_critica();
```

Pode acontecer que as duas threads coloquem o mesmo valor na vez.

O vetor vez pode ser alterado por outrar threads enquanto obtemos o minimo_nao_nulo(vez).

Isso é um problema?

Não, pois nesse caso vamos ter um valor menor que o mínimo atual, e esperaremos mais um pouco no while.

O problema é: Dias threads pegarem o mesmo valor de vez, i.é, vez[i] == vez[j]

Solução: Usar o número da thread para desempatar.

Se tivessemos só duas threads, este laço poderia ser assim:

```
while(vez[i] > vez[outro] || (vez[i] == vez[outro] && i > outro))
  ;
```

Mas temos que pensar no caso em que o outro não está interessado `vez[outro] == 0`).

```
while(vez[i] > minimo_nao_nulo(vez));

while((vez[i] vez[outro]
      || (vez[i] == vez[outro] && i > outro))
      && vez[outro] != 0)
```

Versão 0.1

```
entra_regia_critica(int i) {
    vez[i] = max(vez[i], vez[outro]) + 1;
    while((vez[i] > vez[outro]
          || (vez[i] == vez[outro] && i > outro))
          && vez[outro] != 0)
      ;

sai_regiao_critica(int i) {
    vez[i] = 0;
}
```

Bug com a versão 0.1:

Professor. Dá uma olhada nessa parte seguinte. No cardeno está confuso...

```
thread 0                                thread 1

calcula max = 0                         calcula max = 0
                                        guarda 1 em vez[1]
guarda 1 em vez[0]
                                        avalia condição do while
                                        entra r.c.
avalia condção do while
entra r.c.
```

Versão correta para duas threads.

```
int vez[2] = {0, 0};
boolean tomando_max[2] = {false, false};

entra_regia_critica(int i) {
```

```
    tomando_max[i] = true;
    vez[i] = max(vez[i], vez[outro]) + 1;
    tomando_max[i] = false;
    while(tomando_max[outro])
      ;
    while((vez[i] > vez[outro]
          || (vez[i] == vez[outro] && i > outro))
          && vez[outro] != 0)
      ;
}
```

Bakery para "n" threads.

```
int vez[N] = {0,...,0};
boolean tomando_max[N] = {false,...,false};

entra_regia_critica(int i) {
    vez[i] = max(vez) + 1;
    tomando_max[i] = false;
    for(int j = 0; j < n; j++) {
        if(j != i) {
            while(tomando_max[j]);
            while(vez[j] != 0 && (vez[i] > vez[j]) ||
                 (vez[i] == vez[j] && i > j))
              ;
        }
     }
}
```