Programação Concorrente – Aula 4

Gilmar Gimenes Rodrigues

1 Tie Breaker para n Threads

É uma generalização do tie breaker que resolve o problema da região crítica para n threads, sem ajuda do hardware. O protocolo de entrada tem $n-1$ estágios. Para entrar na região crítica uma thread tem que passar por todos eles. Como cada estágio "segura" a última thread que tentar passar por ele, só uma das n threads poderá passar pelos $n-1$ estágios e entrar na região crítica.

```
int quer_entrar[n] = {-1, -1, ... , -1} /* indexado pela thread */
int ultimo[n] = {-1, -1, ... , -1}      /* indexado pelo estágio */

entra_regiao_critica(int i) { /* i é o número da thread (de 0 a n-1) */
    for (int j = 0; j < n-1; j++) { /* n-1 estágios */
        quer_entrar[i] = j; /* thread i quer passar pelo estágio j */
        ultimo[j] = i;      /* thread i é a última a chegar no estágio j */
        for (int k = 0; k < n; k++) {
            if (i != k) {
                /* espera se alguma thread k já passou pelo estágio j *
                 * e a thread i foi a última a chegar a esse estágio  */
                while (quer_entrar[k] >= quer_entrar[i] && ultimo[j] == i)
                    ; /* busy waiting */
            }
        }
    }
}

sai_regiao_critica(int i) { /* i é o número da thread (de 0 a n-1) */
    quer_entrar[i] = -1;
}
```

Esse algoritmo é $O(n^2)$. Não é eficiente.

2 Ações Atômicas Elementares ou Fine-Grained

São ações atômicas de granularidade fina, implementadas diretamente pelo hardware. Leitura e escrita de words na memória são ações atômicas de granularidade fina.

Ações atômicas corespondem às instruções da máquina?

Não necessariamente. Depende da arquitetura (conjunto de instruções) da máquina. Se a instrução aceita no máximo um operando na memória (`MOV reg, mem` ou `MOV mem, reg`) e se esse operando é acessado (lido ou escito) no máximo uma vez pela instrução, então a instrução é atômica.

Exemplos de instruções que **não** são atômicas:

```
ADD mem, 5
INC mem
DEC mem
SUB mem, 7
```

Pensando em linguagens de alto nível:

- Expressões/atribuições que não referenciam nenhuma variável alterada por outra thread são avaliadas/executadas atômicamente.

```
x = 2 * y + 3 * a[i];
```

- Expressões/atribuições que satisfaçam a propriedade "**at most once**" também são avaliadas/executadas atômicamente.

3 A Propriedade "At Most Once" (A.M.O.)

- Uma expressão **expr** satisfaz a propriedade A.M.O. se a expressão referencia no máximo uma variável simples alterável por outra thread e referencia essa variável no máximo uma vez.
- Uma atribuição **x** = **expr** satisfaz a propriedade se:
  - A **expr** satisfaz e **x** não é lido (acessado) por outra thread.
  - Se **x** é simples (pode ser visto por outra thread) e **expr** não se refere a nenhuma variável alterável por outra thread.

Exemplos:

```
int x, y;

x = y + 1;      /* concorrente com */       y = 5;   /* (satisfaz) */

x = 5;          /* concorrente com */       x = 7;   /* (satisfaz) */
```

O mesmo exemplo com x long:

```
long x, y;

x = 5;          /* concorrente com */       x = 7;   /* (não satisfaz) */
```

Outro exemplo:

```
int x, y;

x = y +1;       /* concorrente com */       y = x + 1;  /* (não satisfaz) */
```

4 Ações Atômicas "Compostas" ou Coarse-Grained

Notação (do livro do Andrews):

- `<x = 1; y = 7;>`

  Indica a ação atômica de "granularidade grossa" formada pelas ações elementares `x = 1` e `y = 7`, executadas de modo indivisível.

- `<await x > y; z = x;>`

  Indica a ação atômica coarse-grained que aguarda a condição `x > y` ser verdadeira e então executa o comando `z = x`.

A forma geral de uma ação atômica com `await` é

```
<await condição; comando(s);>
```

Atenção: a ação atômica e o fragmento de código abaixo não são sinônimos.

```
<await x > y; z = x;>
```

```
while (x <= y)
    ;  /* busy wait */
z = x;
```

No fragmento da direita não é garantido que o valor atribuído a `z` é maior que `y` (outra thread pode alterar `x` antes da atribuição `z = x`).