Token passing algorithms

Guilherme Sousa Lopes - 535869
Lucas Rodrigues Aragão - 538390

Introdução

Motivação

- Para resolver o problema da seção crítica existem diversas soluções diferentes, como uso de monitores e semáforos distribuídos.

- Uma terceira forma é utilizar tokens para passagem de informações.

- A solução é justa e descentralizada, assim como a solução de semáforos, mas nesse caso a troca de mensagens é bem menor.

Formalização

- Sejam *USER[1:n]* uma coleção de processos,
- Para cada processo usuário é criado um processo *Helper*. Os processos *HELPER[1:n]*, formam o anel de comunicação.

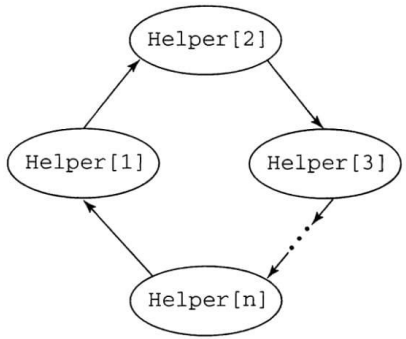

Figure 9.14 A token ring of helper processes.

Solução - Invariante

DMUTEX: `User[i]` is in its CS $\Rightarrow$ `Helper[i]` has the token $\wedge$
there is exactly one token

Solução

```
chan token[1:n](), enter[1:n](), go[1:n](), exit[1:n]();

process Helper[i = 1 to n] {
  while (true) {   # loop invariant DMUTEX
    receive token[i]();          # wait for token
    if (not empty(enter[i])) {   # does user want in?
      receive enter[i]();          # accept enter msg
      send go[i]();                # give permission
      receive exit[i]();           # wait for exit
    }
    send token[i%n + 1]();       # pass token on
  }
}
```

Solução

```
process User[i = 1 to n] {
  while (true) {
    send enter[i]();              # entry protocol
    receive go[i]();
    critical section;
    send exit[i]();               # exit protocol
    noncritical section;
  }
}
```

Detecção de terminação em um anel

Motivação

- Identificar quando um programa distribuído terminou não é simples. Por exemplo: todos os processadores podem estar ociosos e ainda sim existirem mensagens transitando.

- A passagem de token pode ser aplicada para detectar quando uma computação distribuída chegou ao fim.

- Nesse primeiro momento vamos ver a rede de comunicação como um anel, em que um processador só receberá mensagens por um canal próprio e só enviará mensagens para o processador a sua “frente”.

Funcionamento

- CORES:
  - azul -> ocioso / parou
  - vermelho -> ativo
- Todos os processos começam vermelhos. Um processo se torna azul quando recebe o token e volta a ser vermelho ao receber uma mensagem.
- Começa com o token no t[1], deixa ele azul. Após isso, o token percorre todo o anel e, quando voltar ao t[1], se ele ainda estiver azul, temos a garantia de que o processo de fato terminou, concluindo que não há nenhuma mensagem em canais nem processos esperando.

Solução - Invariante

$$DTERM: \text{ every process is idle } \wedge \text{ no messages are in transit}$$

Solução

Global invariant *RING:*

`T[1]` is **blue** $\Rightarrow$ (`T[1]` ... `T[token+1]` are **blue** $\wedge$
`ch[2]` ... `ch[token%n + 1]` are empty)

actions of `T[1]` when it first becomes idle:

```
color[1] = blue; token = 0; send ch[2](token);
```

actions of `T[2]`, ..., `T[n]` upon receiving a regular message:

```
color[i] = red;
```

actions of `T[2]`, ..., `T[n]` upon receiving the token:

```
color[i] = blue; token++; send ch[i%n + 1](token);
```

actions of `T[1]` upon receiving the token:

```
if (color[1] == blue)
```
announce termination and halt;
```
color[1] = blue; token = 0; send ch[2](token);
```

Figure 9.16 Termination detection in a ring.

Detecção de terminação em um grafo

Motivação

- Agora não vamos mais supor que a comunicação ocorre completamente em um anel e sim em um grafo qualquer.

- Assim como no anel, os processadores são os nós e as arestas direcionadas representam os canais de comunicação entre processos.

- Especificamente para a sessão do livro assume-se um grafo completo, ou seja, todos os processos estão interconectados.

Solução

- Estendemos a solução anterior para o caso do grafo completo.
- Devemos garantir que o token passou por todas as arestas do grafo, visitando todos os processos várias vezes.
- Se todos os processos continuaram ociosos desde que começamos a contagem, podemos concluir que a computação terminou.
- Os processos também são coloridos de azul ou vermelho para representar seu estado.

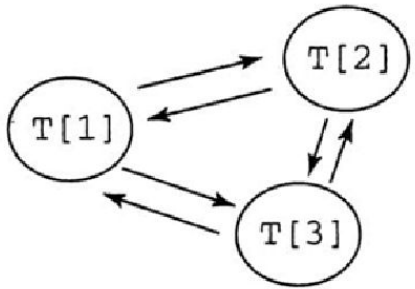

Figure 9.17 A complete communication graph.

Solução

- A contagem aqui é mais complexa.
- Precisamos percorrer o ciclo que passa por todas as arestas.
- Começamos em um ponto qualquer e, a cada vez que o token passa para um processo azul o valor que ele carrega é incrementado.
- Se o token chega em um nó vermelho, o valor é zerado.
- Devemos percorrer todo o ciclo duas vezes, uma vez para tornar todos os processos azuis e outra para conferir se ainda estão.
- Depois disso, podemos garantir que a computação terminou.

Global invariant *GRAPH:*

`token` has value V ⇒
(the last V channels in cycle `C` were empty ∧
the last V processes to receive the token were `blue`)

actions of `T[i]` upon receiving a regular message:

```
color[i] = red;
```

actions of `T[i]` upon receiving the token:

```
if (token == nc)
```
announce termination and halt;
```
if (color[i] == red)
  { color[i] = blue; token = 0; }
else
  token++;
```
set `j` to index of channel for next edge in cycle `C`;
```
send ch[j](token);
```

Figure 9.18 Termination detection in a complete graph.