Sistemas Distribuidos e Tolerância a Falhas

Coordenação de Processos em Sistemas Distribuidos

Trabalho realizado por:
Marisa Quaresma, M1212
Mike Marques, A13863

Coordenação de Processos

A coordenação consiste na interacção entre processos com trocas de mensagens, de forma a obter acesso a zonas de memória partilhadas (buffers) ou recursos partilhados (impressoras, etc).

Produtor-Consumidor

- 2 processos (Produtor e Consumidor)
- 1 bloco de memória comum (Buffer).
- Produtor gera dados (Items) e coloca no Buffer.
- Consumidor retira pela ordem em que foram colocados.
- Buffer permite que variações na velocidade de produção não tenham reflexo directo na velocidade de consumo.
- Produtor é suspenso quando Buffer enche. Da mesma forma, o Consumidor será suspenso quando o Buffer esvazia.

Produtor-Consumidor

Produtor

```
00  while (1) {
01      item = produce();
02      while (n==MAX)
03              ;
04
05      buffer[in]=item;
06      in=(in+1)%MAX;
07          n++;
08
09  }
```

Consumidor

```
20  while (1) {
21              while (!n)
22                  ;
23
24      item=buffer[out];
25
        out=(out+1)%MAX;
26      n--;
27
28       consume(item);
29  }
```

Produtor-Consumidor com Exclusão Mútua

- Problemas:
  - 2 ou mais processos tentam escrever na mesma variável.
  - 1 processo lê um valor que está a ser alterado por outro.

- Solução:
  - Dar acesso exclusivo a cada processo (*Produtor-Consumidor com Exclusão Mútua)*

Produtor-Consumidor com Exclusão Mútua

Produtor

```
00  while (1) {
01      item = produce();
02      while (n==MAX)
03              ;
04
05      begin-mutual-exclusion;
06      buffer[in]=item;
07      in=(in+1)%MAX;
08      n++;
09      end-mutual-exclusion;
10  }
```

Consumidor

```
20  while (1) {
21      while (!n)
22              ;
23
24      begin-mutual-exclusion;
25      item=buffer[out];
26      out=(out+1)%MAX;
27      n--;
28      end-mutual-exclusion;
29      consume(item);
30  }
```

Implementação do Lock

- Variável com 2 estados possiveis (booleano): Open ou Closed
- No máximo, um processo adquire o Lock num determinado momento (lock=CLOSED)
- Apenas o processo que possui Lock executa a secção critica
- Quando termina a execução liberta o Lock (lock=OPEN)
- 2 Formas (Naive e Test-and-set)

Implementações do Lock

1)Naive

```
00   shared:
0 1        LOCK  lock = OPEN;
02
03    begin-mutual-exclusion is
04         while(lock==CLOSED)
05                   ;
06         lock=CLOSED;
07    end;
08
09    end-mutual-exclusion is
1 0        lock=OPEN;
1 1             end;
```

2)Test-and-set

```
00   shared:
0 1        LOCK  lock = OPEN;
02
03    begin-mutual-exclusion is
04         while(test-and-set(lock))
05                   ;
06    end;
07
08
09    end-mutual-exclusion is
1 0        lock=OPEN;
1 1             end;
```

Produtor-Consumidor com Semáforos

- Método mais poderoso
- Sincronização com base em 2 primitivas: Wait e Signal
- Wait decrementa o semáforo
- Signal incrementa o semáforo
- Exclusão mútua (Mutex Locks)

Produtor-Consumidor com Semáforos

Produtor

```
00  while (1) {
01      item = produce();
02      wait (sem-free 1);
03      wait(mutex);
04      buffer[in]=item;
05      in=(in+1)%MAX;
06      signal(mutex);
07      signal(sem-items);
08  }
```

Consumidor

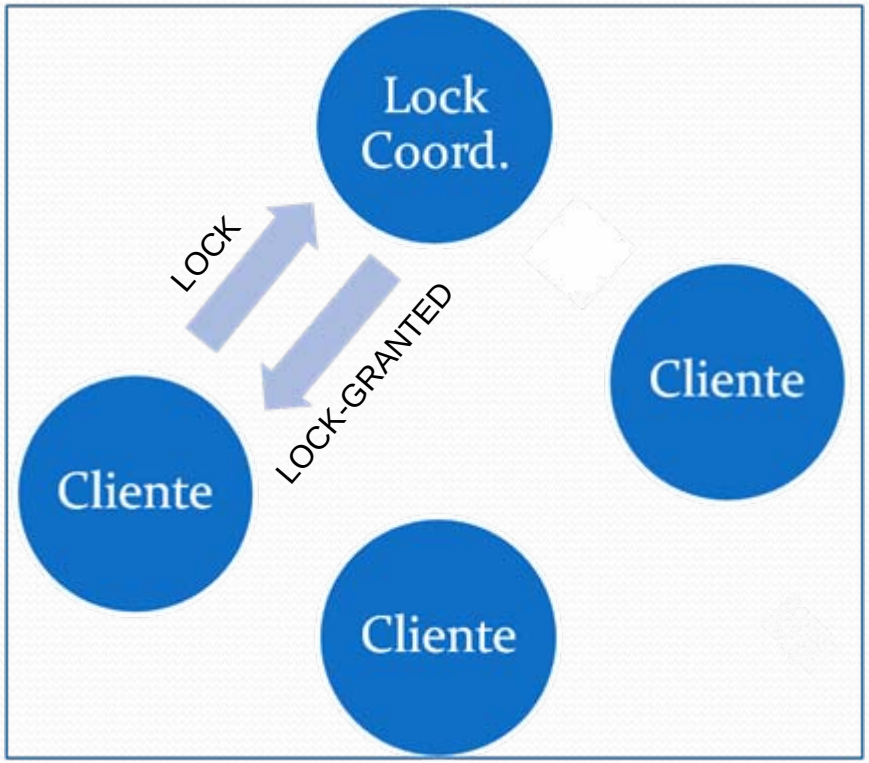
```
20  while (1) {
21      wait (sem-items);
22      wait(mutex);
23      item=buffer[out];
24      out=(out+1)%MAX;
25      signal(mutex);
26      signal(sem-free);
27      consume(item);
28  }
```

Exclusão Mútua Distribuída

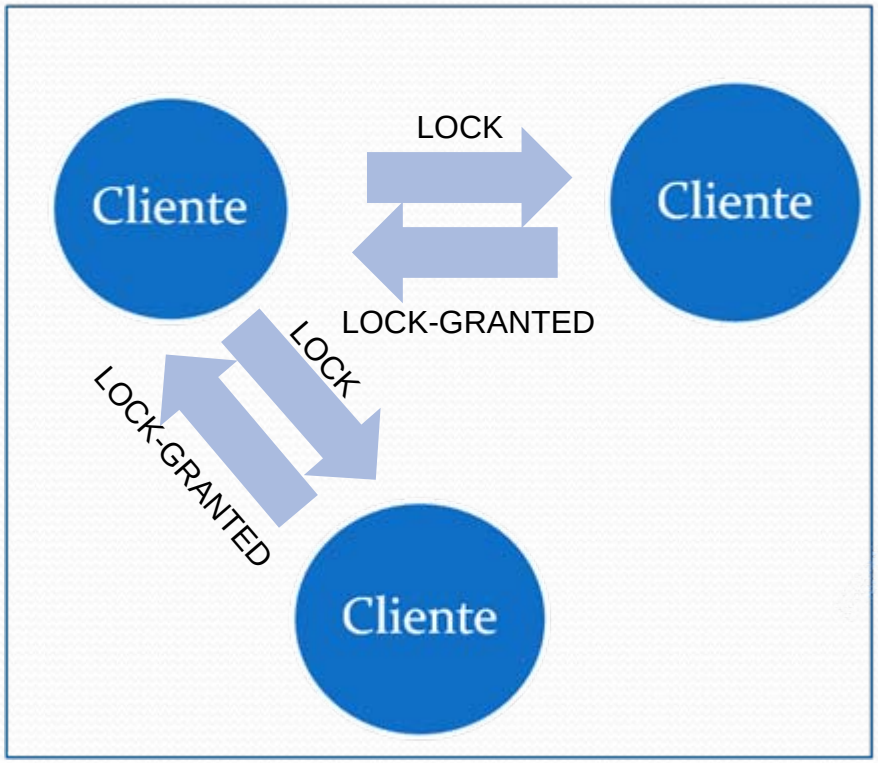
- Participação de todos os processos
- Sincronização do clock (algoritmo de Lamport)
- FIFO e relógios lógicos (algoritmos de ordenação por timestamp)
- Envio de mensagens confirmadas (ACK)

Controlo de Exclusão Mútua Distribuída

Centralizado



Distribuido



Eleição do Líder

- Controlo centralizado, apresentando um servidor de controlo.
- Desvantagem:
  - O sistema fica indisponível quando o servidor “crash”
- Solução:
  - Permitir que qualquer processo assuma o papel de servidor de controlo.
  - Utilizar um algoritmo distribuído de eleição do líder.

Eleição do Líder

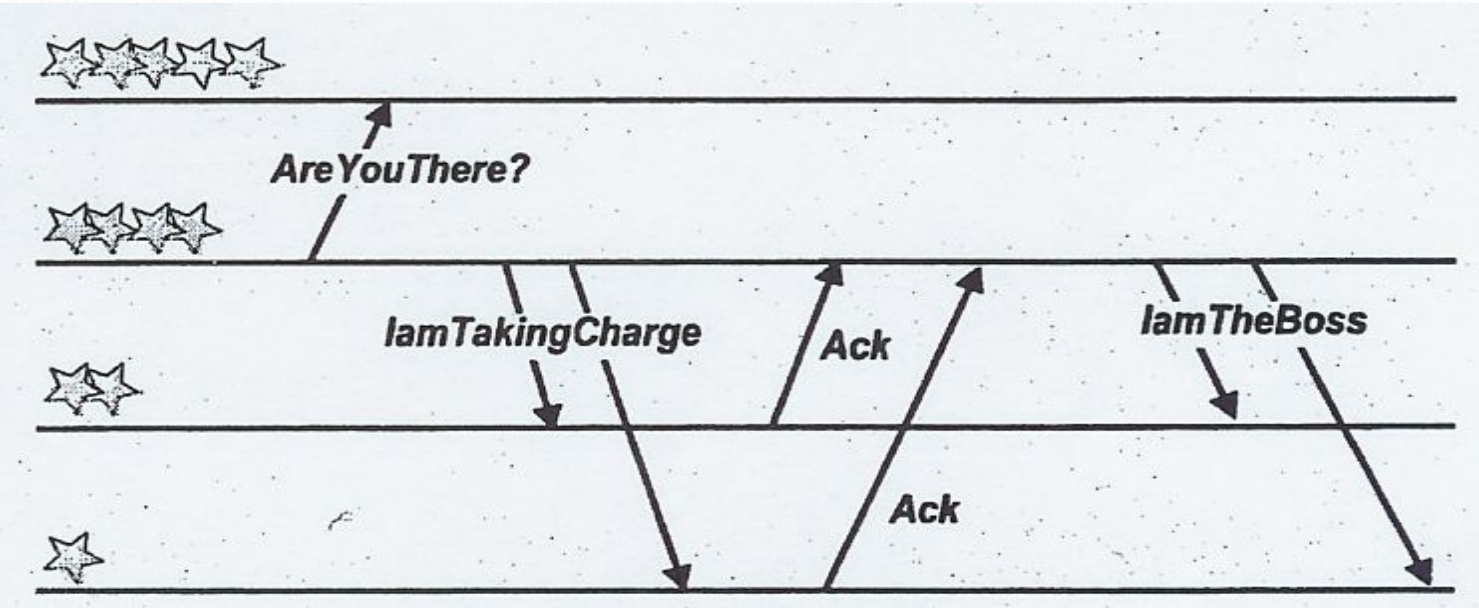
- Semelhanças com algoritmo de exclusão mútua.
- Algoritmo de exclusão mútua:
  - “elege” qual o processo que terá acesso à região crítica
- Algoritmo de eleição do líder:
  - Todos os processos “pedem” o lock;
  - O primeiro processo a adquirir o lock torna-se o líder.

Eleição do Líder

- Optimizações:
  - Um processo pode abortar a sua execução assim que o líder for escolhido.
  - Se um processo recebe um pedido de lock de outro processo p, este poderá suportar p eleições, em vez de, competir para se tornar líder.
- Algoritmo de Bully:
  - Assume que um processo sabe a prioridade de todos os outros no sistema.
  - Vence o processo que tem maior rank.

Eleição do Líder

- Algoritmo de Bully:



Deadlock

- Ocorre quando:
  - dois ou mais processos esperam uns pelos outros numa configuração onde nenhum progresso pode ser feito.
  - Em sistemas centralizados, onde os processos precisam de ser sincronizados.

Deadlock

Produtor

```
00  while (1) {
01      item = produce();
02      wait (mutex);
03      wait(sem-free,1);
04      buffer[in]=item;
05      in=(in+1)%MAX;
06      signal(mutex);
07      signal(sem-items);
08  }
```

Consumidor

```
20  while (1) {
21      wait(sem-items);
22      wait(mutex);
23      item=buffer[out];
24      out=(out+1)%MAX;
25      signal(mutex);
26      signal(sem-free);
27       consume(item);
28  }
```

Deadlock

- Deadlock ocorre sempre que as seguintes quatro condições estejam presentes:
  - Exclusão mútua:
    - Alguns recursos não são partilhados e apenas podem ser acedidas por um processo de cada vez.
  - Hold-and-wait:
    - Pelo menos um processo espera por recursos adicionais enquanto contém recursos não partilhados.
  - Sem preempção:
    - Um processo que contém um recurso é lhe permitido continuar a mantê-lo até estar pronto para o libertar.

Deadlock

- Circular-wait:
  - Existe uma cadeia circular de n processos, onde p0 se encontra à espera de um recurso que p1 tem, que por sua vez se encontra à espera de um recurso mantido por p2…

- Solução:
  - Eliminar uma destas condições (Certo ou Errado?).

Deadlock

- Porque não é fácil eliminar uma destas condições?
  - A exclusão mútua poderá ser eliminada:
    - se existirem apenas recursos partilhados, mas isto é uma condição muito restritiva para a maioria das aplicações:
      - ex: estruturas de dados partilhados são simplesmente não partilhadas
  - Hold-and-wait poderá ser eliminada:
    - Se um processo apenas manter um recurso de cada vez, mas mais uma vez é uma condição muito restritiva para a maioria das aplicações.

Deadlock

- Alternativa:
  - Forçar um processo a pedir à priori todos os recursos que vai precisar.
  - Desvantagens:
    - Ineficiente- alguns recursos são locked mesmo antes de serem necessários.
- Não-Preempção pode ser eliminada:
  - Permitindo a preempção, ou seja, forçando um processo a libertar os seus recursos.
- Circular-wait pode ser eliminada:
  - Através da imposição de uma ordem a todos os recursos, forçando os processos a adquirir os recursos pela mesma ordem.

Deadlock

- Alternativas à prevenção do deadlock:
  - Detecção e resolução do deadlock:
    - Consiste em automatizar o processo para destruir o deadlock, fazendo com que um ou mais processos abortem.
  - Evitar o deadlock (deadlock avoidance):
    - Consiste em fazer checks antes de um recurso ser dado a um processo, para se ter a certeza que as condições para o deadlock não ocorrem.