



Programação multi-tarefa em Memória Partilhada

Parte II – Programação avançada

Sistemas Operativos



Programar com objetos partilhados

Programar com objetos partilhados

Hoje
vamos
discutir
esta parte
em maior
detalhe

- Até agora, aprendemos esta receita para programar concorrentemente com memória partilhada:
 - Identificar secções críticas
 - Sincronizar cada secção crítica com trinco (*mutex*)



Aspetos avançados para discutir hoje

- Um trinco global ou múltiplos trincos finos?
- Preciso mesmo usar trinco?

Como sincronizar esta função?

```
struct {
    int saldo;
    ...
} conta_t;

int levantar_dinheiro (conta_t* conta, int valor) {

    mutex_lock(____);
    if (conta->saldo >= valor)
        conta->saldo = conta->saldo - valor;
    else
        valor = -1; /* -1 indica erro ocorrido */

    mutex_unlock(____);
    return valor;
}
```

Trinco global

- Normalmente é a solução mais simples
- Mas limita o paralelismo
 - Quanto mais paralelo for o programa, maior é a limitação
- Exemplo: “big kernel lock” do Linux
 - Criado nas primeiras versões do Linux (versão 2.0)
 - Grande barreira de escalabilidade
 - Finalmente removido na versão 2.6

Trincos finos: programação com objetos partilhados

- Objeto cujos métodos podem ser chamados em concorrência por diferentes tarefas
- Devem ter:
 - Interface dos métodos públicos
 - Código de cada método
 - Variáveis de estado
 - **Variáveis de sincronização**
 - Um trinco para garantir que métodos críticos se executam em exclusão mútua
 - Opcionalmente: semáforos, variáveis de condição
 - Estas abstracções serão introduzidas nas próximas aulas

Programar com trincos finos

- Em geral, maior paralelismo
- **Mas pode trazer *bugs* difíceis de resolver...**

Exemplo com trincos finos

```
transferir(conta a, conta b, int montante) {  
    fechar(a.trinco);  
    debitar(a, montante);  
    fechar(b.trinco);  
    creditar(b, montante);  
    abrir(a.trinco);  
    abrir(b.trinco);  
}
```

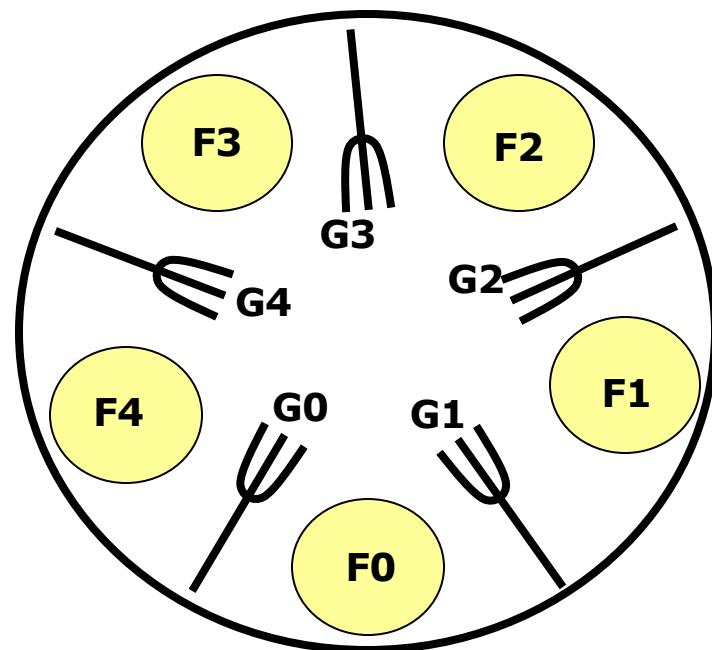
- O que pode correr mal?

Jantar dos Filósofos

- Cinco Filósofos estão reunidos para filosofar e jantar spaghetti:
 - Para comer precisam de dois garfos, mas a mesa apenas tem um garfo por pessoa.
- Condições:
 - Os filósofos podem estar em um de três estados : Pensar; Decidir comer ; Comer.
 - O lugar de cada filósofo é fixo.
 - Um filósofo apenas pode utilizar os garfos imediatamente à sua esquerda e direita.



Jantar dos Filósofos



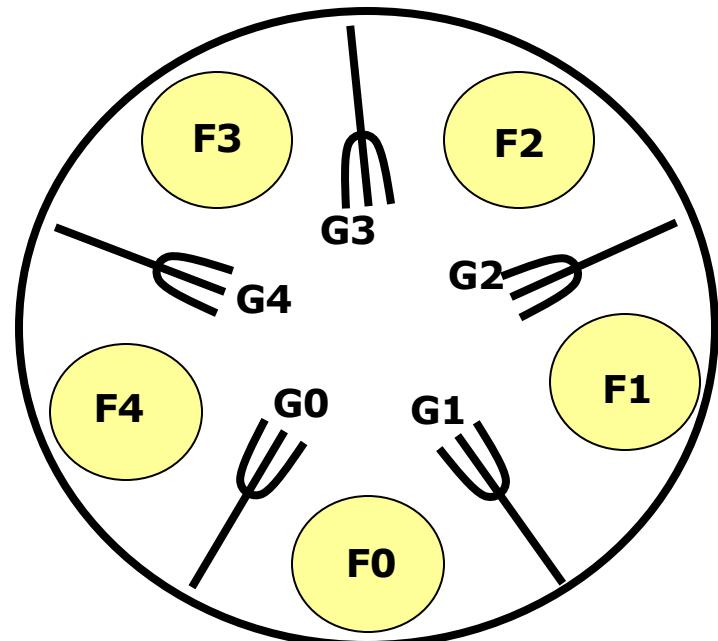


Jantar dos Filósofos

```
filosofo(int id) {  
    while (TRUE) {  
        pensar();  
        <adquirir os garfos>  
        comer();  
        <libertar os garfos>  
    }  
}
```

Jantar dos Filósofos, versão #1

```
mutex_t garfo[5] = {...};  
  
filosofo(int id)  
{  
    while (TRUE) {  
        pensar();  
        fechar(garfo[id]);  
        fechar(garfo[(id+1)%5]);  
        comer();  
        abrir(garfo[id]);  
        abrir(garfo[(id+1)%5]);  
    }  
}
```



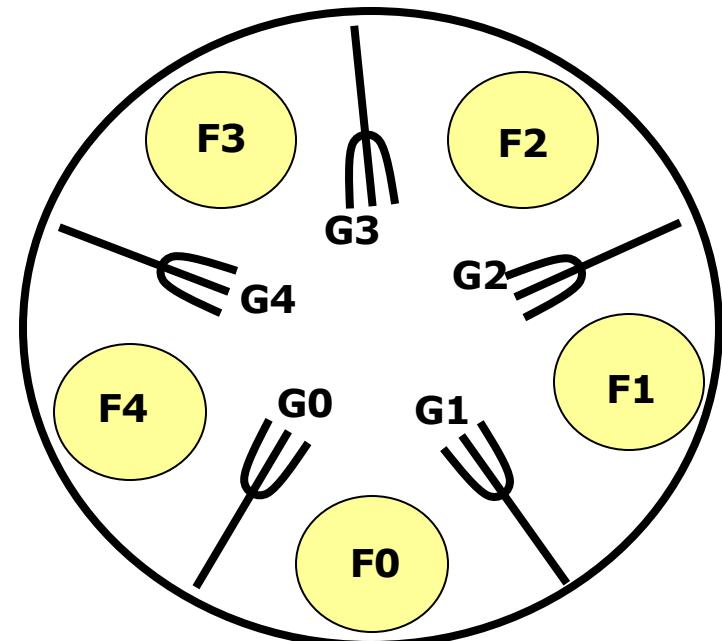
Problema?



Como prevenir interblocagem?

Prevenir a interblocagem: uma solução

```
mutex_t garfo[5] = {...};  
  
filosofo(int id)  
{  
    while (TRUE) {  
        pensar();  
        if (id < 4) {  
            fechar(garfo[id]);  
            fechar(garfo[(id+1)%5]);  
        } else {  
            fechar(garfo[(id+1)%5]);  
            fechar(garfo[id]);  
        }  
        comer();  
        abrir(garfo[id]);  
        abrir(garfo[(id+1)%5]);  
    }  
}
```



Princípio base: garantir que os recursos são todos adquiridos segundo uma ordem total pré-definida

Prevenir a interblocagem: outra solução

```
mutex_t garfo[5] = {...};  
  
filosofo(int id)  
{  
    int garfos;  
    while (TRUE) {  
        pensar();  
        garfos = FALSE;  
        while (!garfos) {  
            if (lock(garfo[id])  
                if (try_lock(garfo[(id+1)%5])  
                    garfos = TRUE;  
                else { // adquisição 2º trinco falhou  
                    unlock(garfo[id]); // abre 1ºtrinco e tenta outra vez  
                }  
        }  
        comer();  
        unlock(garfo[id]);  
        unlock(garfo[(id+1)%5]);  
    }  
}
```

**Resolvemos o problema
da interblocagem!
...e o da míngua?**

Evitar míngua: recuo aleatório!

- Pretende-se evitar que dois filósofos vizinhos possam conflituar indefinidamente



Introduzir uma fase de espera/recuo (*back-off*) entre uma tentativa e outra de cada filósofo.

- Como escolher a duração da fase de espera?
 - Inúmeras políticas propostas na literatura
 - Vamos ilustrar apenas os princípios fundamentais das políticas mais genéricas e simples

Evitar míngua: escolha da duração do recuo

- Duração aleatória entre [0, MAX]
 - Porquê aleatória?
- Como escolher o valor MAX?
 - Quanto maior o valor de MAX:
 - menor desempenho
 - maior probabilidade de evitar contenção
 - Adaptar o valor de MAX consoante o número de tentativas
 - Por exemplo, $\text{MAX} = \text{constante} * \text{num_tentativas}$

Políticas de recuo: variantes

- Ter em conta a prioridade da tarefa na definição do período máximo de recuo
 - Tarefas mais prioritárias esperam menos
- Adaptar a duração máxima ao número máximo de tarefas concorrentemente ativas
- Adaptar o período de espera à duração prevista das tarefas com as quais se entra em contenção:
 - as tarefas devem anunciar o progresso atingido ao executar na secção critica

Prevenir a interblocagem: récuo aleatório

```
mutex_t garfo[5] = {...};  
  
filosofo(int id)  
{  int garfos;  
    while (TRUE) {  
        pensar();  
        garfos = FALSE;  
        while (!garfos) {  
            if (lock(garfo[id])  
                if (try_lock(garfo[(id+1)%5])  
                    garfos = TRUE;  
                else { // adquisição 2º trinco falhou  
                    unlock(garfo[id]); // abre 1ºtrinco e tenta outra vez  
                    sleep(random([0, MAX]);  
                }  
            }  
            comer();  
            unlock(garfo[id]);  
            unlock(garfo[(id+1)%5]);  
        }  
    }  
}
```



Recapitulando: como prevenir interblocagem?

- Garantir que os recursos são todos adquiridos segundo uma ordem total pré-definida
- Quando a aquisição de um recurso não é possível, libertando todos os recursos detidos e anulando as operações realizadas até esse momento

Vantagens/desvantagens de cada abordagem?