PROBLEMA DO PRODUTOR CONSUMIDOR

- Este problema considera dois tipos de processos:
- **Produtor**:
  - Cria elementos de dados e insere em um *buffer*, se houver espaço. Caso não haja, o produtor deve ser bloqueado
- **Consumidor**:
  - Retira elementos do *buffer*, se houver, e executa uma determinada tarefa com eles. Se não houver elementos no *buffer*, deve ser bloqueado.

- Um semáforo pode ser usado controlar os consumidores, indicando quantos elementos estão no *buffer*.
    - Quando a quantidade for zero, novas tentativas de obter um elemento bloqueiam o consumidor.
- O bloqueio do produtor deve ocorrer no tamanho máximo do *buffer*, característica inexistente em um semáforo.
    - Usamos um outro semáforo que controle as **posições vazias** do *buffer*
    - Assim, quando o *buffer* estiver cheio, o semáforo estará zerado, bloqueando novos produtores

```
semáforo naoVazio = 0;

semáforo naoCheio = N;  // tamanho do buffer

      Produtor                              Consumidor

Laço infinito                         Laço infinito
    d=produz_dados()                      wait(naoVazio)
    wait(naoCheio)                        d=Le_dados(buffer)
    Adiciona(buffer, d)                   signal(naoCheio)
    signal(naoVazio)                      Usa_dados(d)
Fim_laço                              Fim_laço
```

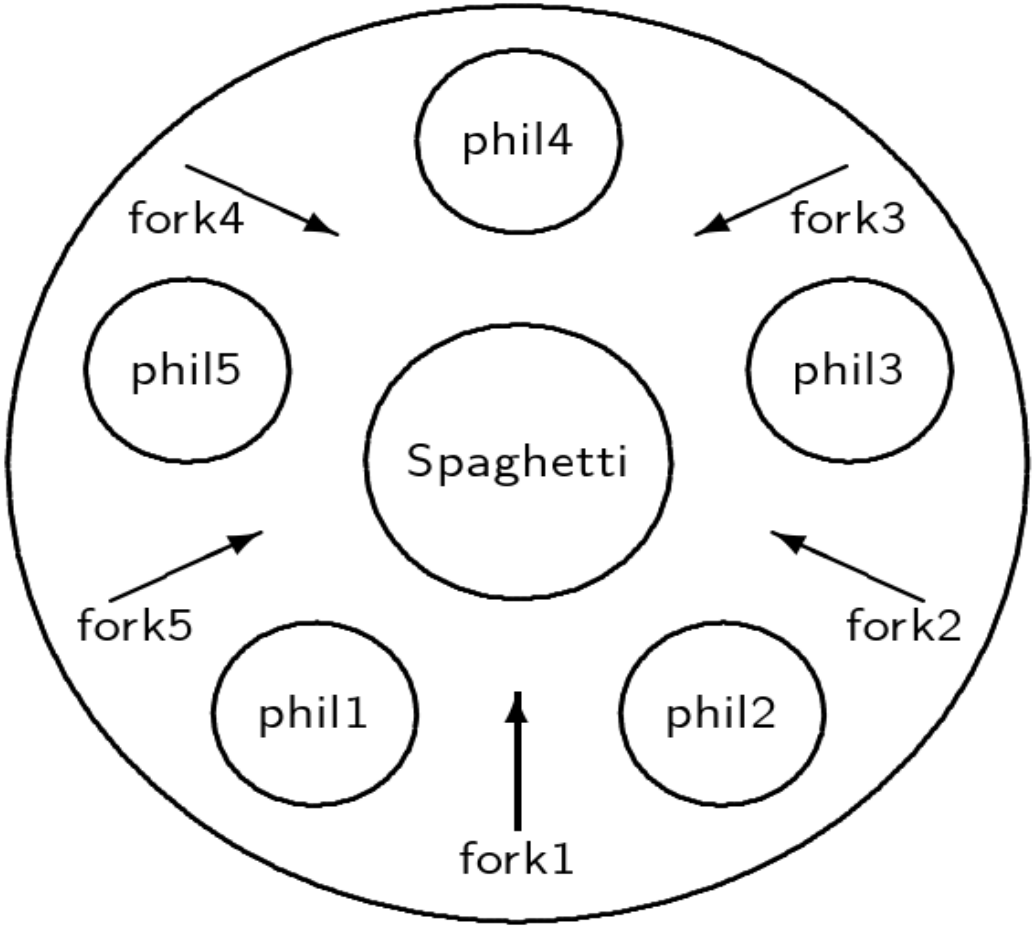
PROBLEMA DO JANTAR DOS FILOSÓFOS

- Serve como critério para comparação entre as diversas técnicas de sincronização, pois:
  - É suficientemente simples
  - Apresenta a situação de disputa por recursos compartilhados, comum em Programação Concorrente

Descrição do Problema:

- Há cinco filósofos que fazem duas atividades: pensam ou comem.
- Os filósofos estão sentados ao redor de uma mesa redonda
- A frente de cada filósofo tem um prato de comida
- Entre os pratos há um garfo. Portanto cada filósofo tem um garfo a sua direita e a sua esquerda, que são compartilhados com filósofos vizinhos
- Para comer, o filósofo deve pegar antes ambos os garfos

- Após terminar de comer, os garfos devem ser devolvidos
- Cada o filósofo deve alternar-se entre pensar e comer



- Qualquer **solução para o problema** requer que:
  - Um filósofo só come se tiver os dois garfos
  - **Exclusão mútua**: Filósofos distintos não podem usar o mesmo garfo ao mesmo tempo
  - **Livre de *deadlock***: Os filósofos devem comer
  - **Livre de *starvation***: Nenhum filósofo deve esperar indefinidamente para comer
  - O maior número de filósofos deve poder comer ao mesmo tempo

- **Tentativa de solução :** Um semáforo para cada garfo

```
semáforo garfo[5] = {1,1,1,1,1}
        Filósofo i
Laço infinito
    Pensando()
    wait(garfo[i])
    wait(garfo[i+1])
    Comendo()
    signal(garfo[i])
    signal(garfo[i+1])
Fim_laço
```

- Esta solução não garante ser livre de *deadlock*, pois:
  - Considere o cenário em que todos filósofos vizinhos "pegam" os respectivos garfos da direita
  - Nenhum deles conseguirá obter o garfo esquerdo

- **Primeira solução:** Limitar o número de filósofos comendo
  - Evita o deadlock, pois caso todos peguem o primeiro garfo, sempre haverá um em condições de pegar o segundo garfo

semáforo comer=4, garfo[5] = {1,1,1,1,1}

Filósofo i: Laço infinito

Pensando()

wait(comer);

wait(garfo[i])

wait(garfo[i+1])

Comendo()

signal(garfo[i])

signal(garfo[i+1])

signal(comer)

Fim_laço

- **Segunda solução**: Inverter a ordem dos garfos de um filósofo

semáforo garfo[5] = {1,1,1,1,1}

Filósofo i (1<=i<=4)	**Filósofo 5**
Laço infinito	Laço infinito
Pensando()	Pensando()
wait(garfo[i])	wait(garfo[i+1])
wait(garfo[i+1])	wait(garfo[i])
Comendo()	Comendo()
signal(garfo[i])	signal(garfo[i+1])
signal(garfo[i+1])	signal(garfo[i])
Fim_laço	Fim_laço

PROBLEMA DOS LEITORES E ESCRITORES

- Este problema simula o acesso concorrente a um banco de dados
- Há dois tipos de processos: Leitores e Escritores

- **Leitor**:
  - Efetua leitura das informações sem alterá-las.
  - Pode acessar os dados concorrentemente com outros leitores.
  - Não pode acessar o banco com um escritor.

- **Escritor**:
  - Altera as informações
  - Deve ter acesso exclusivo ao banco de dados, sem outros leitores ou escritores.

```
Monitor RW
int leitores=0, escritores=0
var_cond Ler, Escrever

função IniciaLeitor()
   Se escritores<> 0 ou nãoVazio(Escrever) então waitC(Ler)
   leitores = leitores+1
   signalC(Ler)

função TerminaLeitor()
   leitores = leitores – 1
   Se leitores == 0 então signalC(Escrever)
```

função IniciaEscritor()

Se escritores<> 0 ou leitores <> 0 então **waitC**(Escrever)

escritores = escritores+1

função TerminaEscritor()

escritores = escritores – 1

Se Vazio(Leitor) então **signalC**(Escrever)

senão **signalC**(Ler)

Leitor:	**Escritor:**
IniciaLeitor()	IniciaEscritor()
//lê a base de dados	//Escreve na base de dados
TerminaLeitor()	TerminaEscritor()