

Variáveis de Condição

Sistemas Distribuídos

Variáveis de condição

- permitem que threads suspendam/retomem a sua execução dentro de zonas críticas, de acordo com uma dada condição
- métodos: **wait()**, **notify()**, **notifyAll()**
- estão associadas a um monitor/lock; cada objecto tem uma variável de condição intrínseca (**wait-set**)

Exemplo:

T1

```
synchronized metodo1(){  
    (...)  
    while(obj.x == 0){  
        obj.wait();  
    }  
    obj.x+=1;  
}
```

T2

```
synchronized metodo2(){  
    (...)  
    while(obj.x == 0){  
        obj.wait();  
    }  
    obj.x+=2;  
}
```

T3

```
synchronized metodo3(){  
    (...)  
    obj.x = 1;  
    obj.notifyAll();  
}
```

obj

x = 0	
Monitor:	<i>em espera p/ monitor:</i>
Wait-Set:	

Exemplo:

T1

```
synchronized metodo1(){  
    (...)  
    while(obj.x == 0){  
        obj.wait();  
    }  
    obj.x+=1;  
}
```

T2

```
synchronized metodo2(){  
    (...)  
    while(obj.x == 0){  
        obj.wait();  
    }  
    obj.x+=2;  
}
```

T3

```
synchronized metodo3(){  
    (...)  
    obj.x = 1;  
    obj.notifyAll();  
}
```

obj

x = 0

Monitor: T1 em espera p/ monitor: T2,T3
Wait-Set:

Exemplo:

T1

```
synchronized metodo1(){  
    (...)  
    while(obj.x == 0){  
        obj.wait();  
    }  
    obj.x+=1;  
}
```

T2

```
synchronized metodo2(){  
    (...)  
    while(obj.x == 0){  
        obj.wait();  
    }  
    obj.x+=2;  
}
```

T3

```
synchronized metodo3(){  
    (...)  
    obj.x = 1;  
    obj.notifyAll();  
}
```

obj

x = 0

Monitor: *em espera p/ monitor: T2,T3*
Wait-Set: **T1**

Exemplo:

T1

```
synchronized metodo1(){  
    (...)  
    while(obj.x == 0){  
        obj.wait();  
    }  
    obj.x+=1;  
}
```

T2

```
synchronized metodo2(){  
    (...)  
    while(obj.x == 0){  
        obj.wait();  
    }  
    obj.x+=2;  
}
```

T3

```
synchronized metodo3(){  
    (...)  
    obj.x = 1;  
    obj.notifyAll();  
}
```

obj

x = 0

Monitor: T2 *em espera p/ monitor: T3*
Wait-Set: T1

Exemplo:

T1

```
synchronized metodo1(){  
    (...)  
    while(obj.x == 0){  
        obj.wait();  
    }  
    obj.x+=1;  
}
```

T2

```
synchronized metodo2(){  
    (...)  
    while(obj.x == 0){  
        obj.wait();  
    }  
    obj.x+=2;  
}
```

T3

```
synchronized metodo3(){  
    (...)  
    obj.x = 1;  
    obj.notifyAll();  
}
```

obj

x = 0

Monitor: *em espera p/ monitor: T3*
Wait-Set: T1, **T2**

Exemplo:

T1

```
synchronized metodo1(){  
    (...)  
    while(obj.x == 0){  
        obj.wait();  
    }  
    obj.x+=1;  
}
```

T2

```
synchronized metodo2(){  
    (...)  
    while(obj.x == 0){  
        obj.wait();  
    }  
    obj.x+=2;  
}
```

T3

```
synchronized metodo3(){  
    (...)  
    obj.x = 1;  
    obj.notifyAll();  
}
```

obj

x = 1

Monitor: T3 *em espera p/ monitor:*
Wait-Set: T1,T2

Exemplo:

T1

```
synchronized metodo1(){  
    (...)  
    while(obj.x == 0){  
        obj.wait();  
    }  
    obj.x+=1;  
}
```

T2

```
synchronized metodo2(){  
    (...)  
    while(obj.x == 0){  
        obj.wait();  
    }  
    obj.x+=2;  
}
```

T3

```
synchronized metodo3(){  
    (...)  
    obj.x = 1;  
    obj.notifyAll();  
}
```

obj

x = 1

Monitor: T3 *em espera p/ monitor:*
Wait-Set: T1, T2

Exemplo:

T1

```
synchronized metodo1(){  
    (...)  
    while(obj.x == 0){  
        obj.wait();  
    }  
    obj.x+=1;  
}
```

T2

```
synchronized metodo2(){  
    (...)  
    while(obj.x == 0){  
        obj.wait();  
    }  
    obj.x+=2;  
}
```

T3

```
synchronized metodo3(){  
    (...)  
    obj.x = 1;  
    obj.notifyAll();  
}
```

obj

x = 3

Monitor: T2 *em espera p/ monitor: T1*
Wait-Set:

Exemplo:

T1

```
synchronized metodo1(){  
    (...)  
    while(obj.x == 0){  
        obj.wait();  
    }  
    obj.x+=1;  
}
```

T2

```
synchronized metodo2(){  
    (...)  
    while(obj.x == 0){  
        obj.wait();  
    }  
    obj.x+=2;  
}
```

T3

```
synchronized metodo3(){  
    (...)  
    obj.x = 1;  
    obj.notifyAll();  
}
```

obj

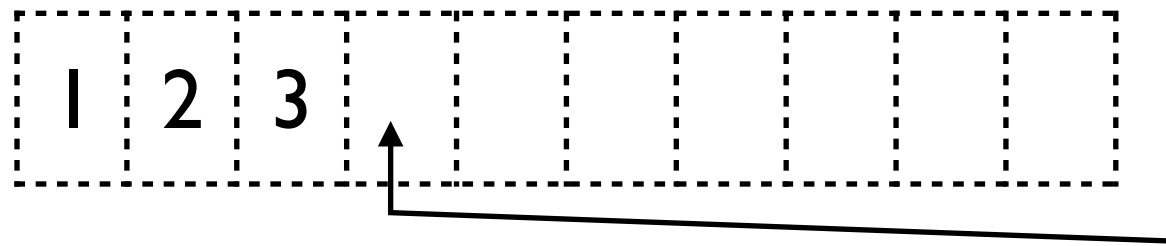
x = 4

Monitor: T1 *em espera p/ monitor:*
Wait-Set:

Exercícios

1) Implemente uma classe **BoundedBuffer** que ofereça as operações **void put(int v)** e **int get()** sobre um array cujo tamanho é definido no momento da construção de uma instância. O método **put** deverá bloquear enquanto o array estiver cheio e o método **get** deverá bloquear enquanto o array estiver vazio. Os métodos oferecidos podem estar sujeitas a invocações de threads concorrentes sobre uma instância partilhada. A classe **BoundedBuffer** deverá garantir a correcta execução em cenário multi-thread.

Exercícios



BoundedBuffer

(LIFO - Last In First Out)

```
private int[] values  
int poswrite;
```

Condições a respeitar:

- **put(v)**: Bloquear enquanto o array estiver cheio;
- **get()**: Bloquear enquanto o array estiver vazio.

Usando *wait* e *notify/notifyAll* e exclusão mútua;

```
void put(int v); ← Produtor  
int get();       implements Runnable  
                ↑  
Consumidor  
implements Runnable
```

Teste:

- BoundedBuffer com 10 posições
- Consumidor invoca get 20x
- Produtor invoca put 20x

Exercícios

2) Considere um cenário produtor/consumidor sobre o **BoundedBuffer** do exercício anterior, com **P** produtores e **C** consumidores, com um número total de threads **C + P = N** e tempos de produção e consumo **T_p** e **T_c**. Obtenha experimentalmente o número óptimo de threads de cada tipo a utilizar para maximizar o débito.

Exercício 2

Cenário de Teste:

$T_c = 0.5\text{seg}$ \leftarrow *tempo de espera entre cada get (tempo de consumo)*

$T_p = 1\text{seg}$ \leftarrow *tempo de espera entre cada put (tempo de produção)*

$\text{totalOps} = 100$ \leftarrow *no total, produzir 100 items e consumir 100 items*

$N = C + P = 10$ \leftarrow *total de threads*

$C = ?$

$P = ?$

Objectivo:

Descobrir C e P de forma a maximizar o débito (ops/seg):

$$\text{débito} = \frac{\text{totalOps}}{(\text{tempoFim} - \text{tempoInicio})}$$

Exercícios

3) Implemente uma classe **Barreira** que ofereça um método **esperar()** cujo objectivo é garantir que cada thread que o invoque se bloqueie até que o número de threads nesta situação tenha atingido o valor **N** passado ao construtor de uma sua instância. Um objecto **Barreira** deverá poder ser reutilizado em sucessivas invocações de **esperar()**.

