UNIP
UNIVERSIDADE PAULISTA

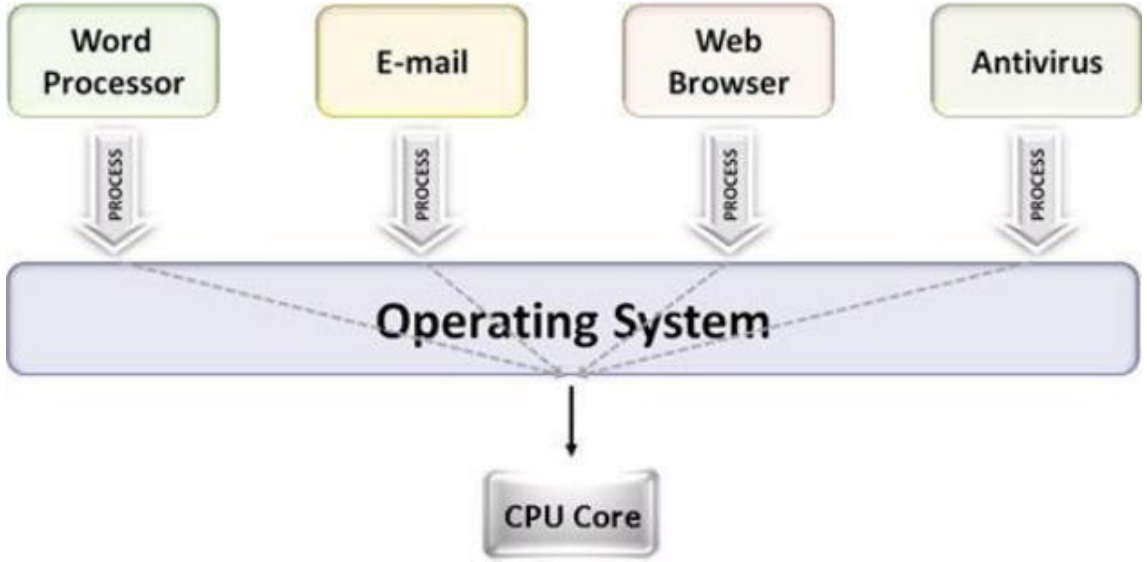
UNICLUSTER

Threads

Carlos Arruda Baltazar
UNIP – Cidade Universitária

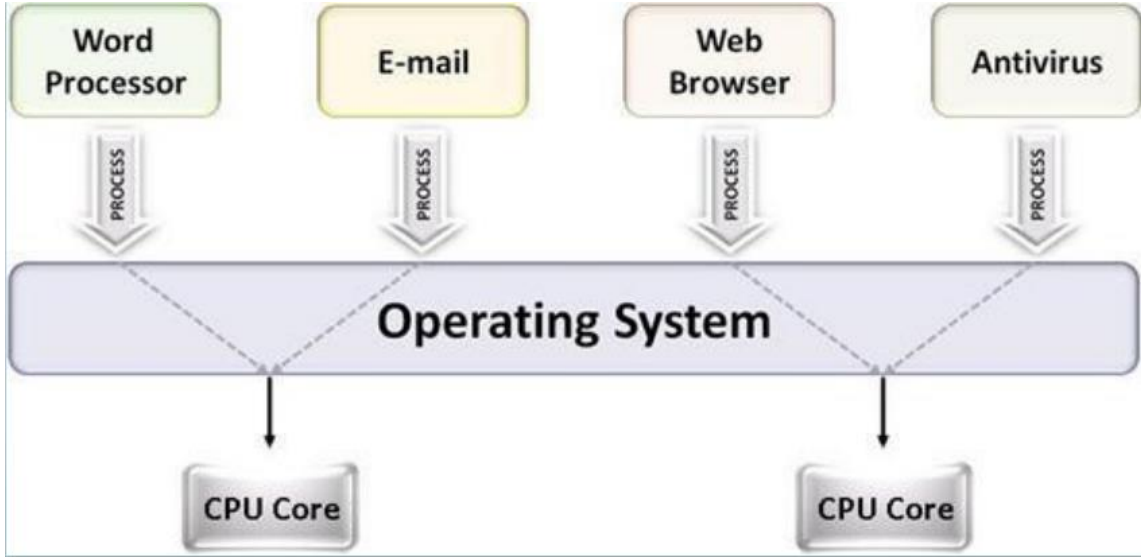
Processos

O processamento de hoje ainda é feito na maioria das vezes por um único processador da máquina.



Processos

Contudo em máquinas com processadores que possuem mais de um core, os processos são escalonados nos diferentes cores do processador.



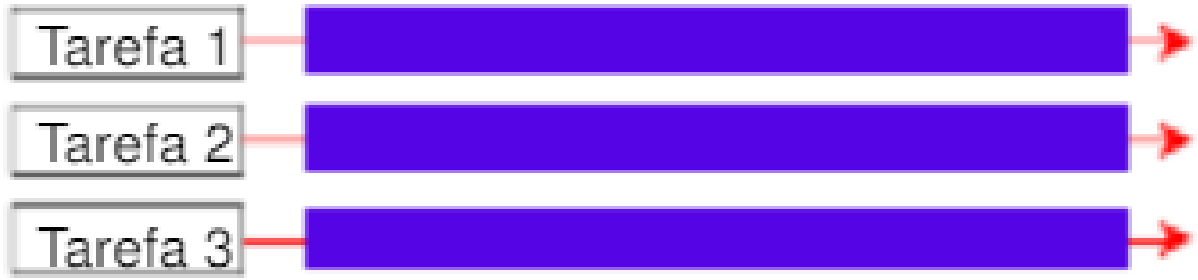
Processos

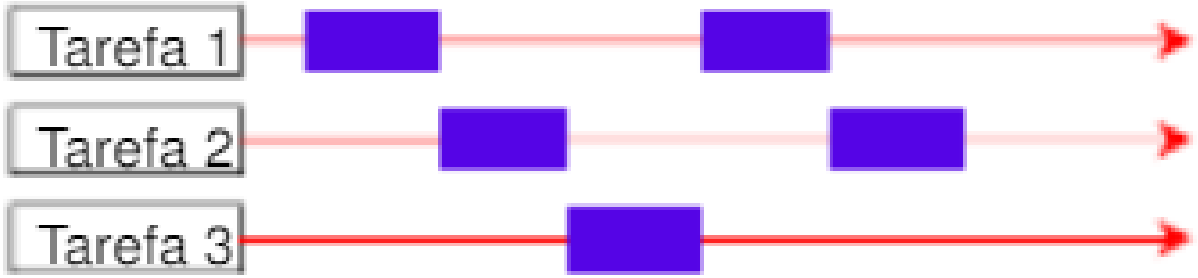
Só que na maioria dos sistemas, vários processos são executados no mesmo intervalo de tempo, dando a impressão de simultaneidade de execução. Esta capacidade é chamada de multiprocessamento, e muito estudada em sistemas operacionais. Um processo é basicamente um programa em execução.

Threads

São subprocessos no Sistema Operacional. A thread pode ser vista como uma parte de um processo, que permite compartilhar a sua área de dados com o programa ou outros threads. Seu início de execução é muito mais rápido do que um processo, e o acesso a sua área de dados funciona como um único programa.

Threads





Vantagens

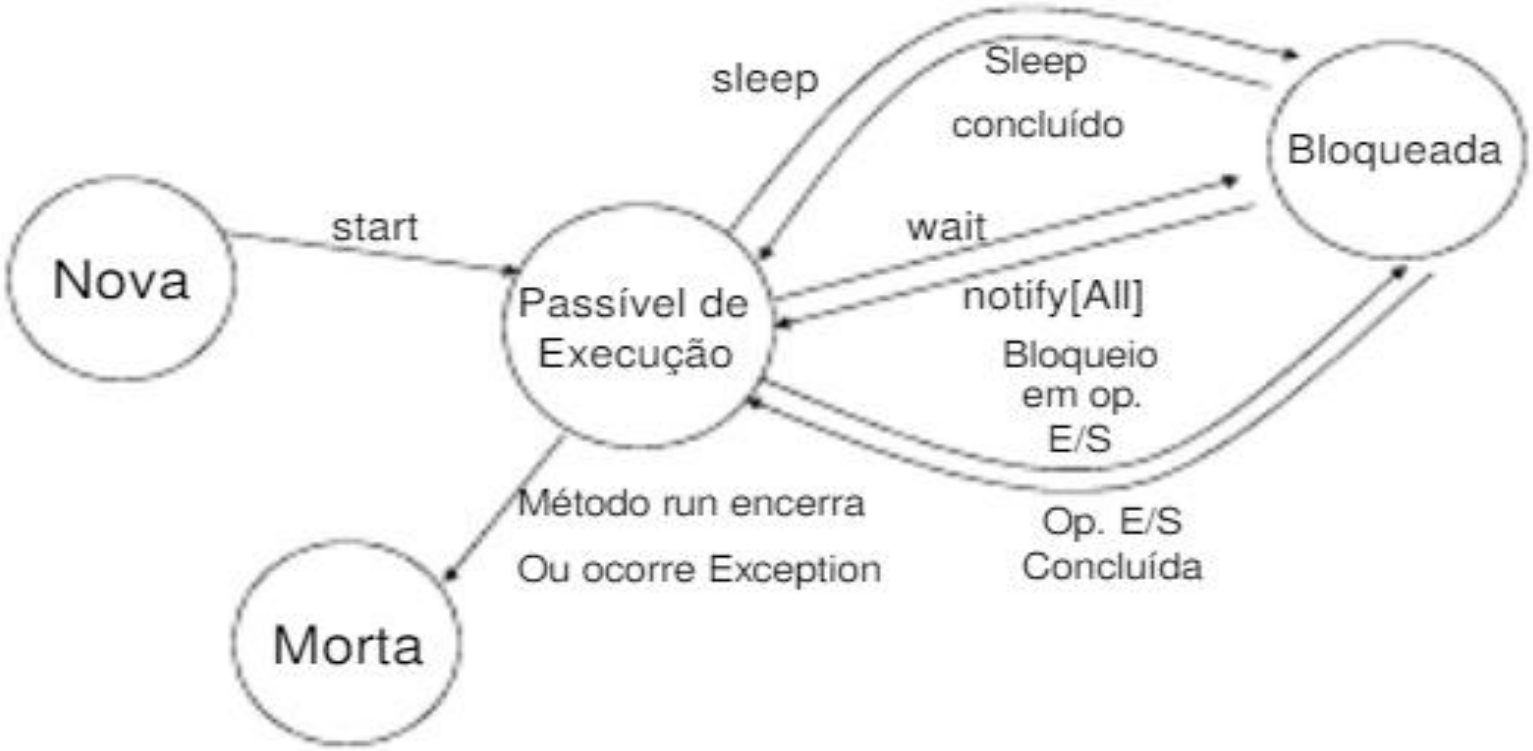
- Torna-se menos custoso em termos de processamento criar um thread que um processo;
- São necessárias poucas variáveis para criação de um thread;
- Podem compartilhar o mesmo espaço de processo;
- Podem compartilhar os mesmos recursos de seus processos.

Estados

- Threads novas: Threads que foram instanciadas e ainda não executadas.

- Threads executáveis ou passíveis de execução: Assim que a thread é iniciada, ela entra neste estado, o que não significa que ela já esteja em execução.

Estados

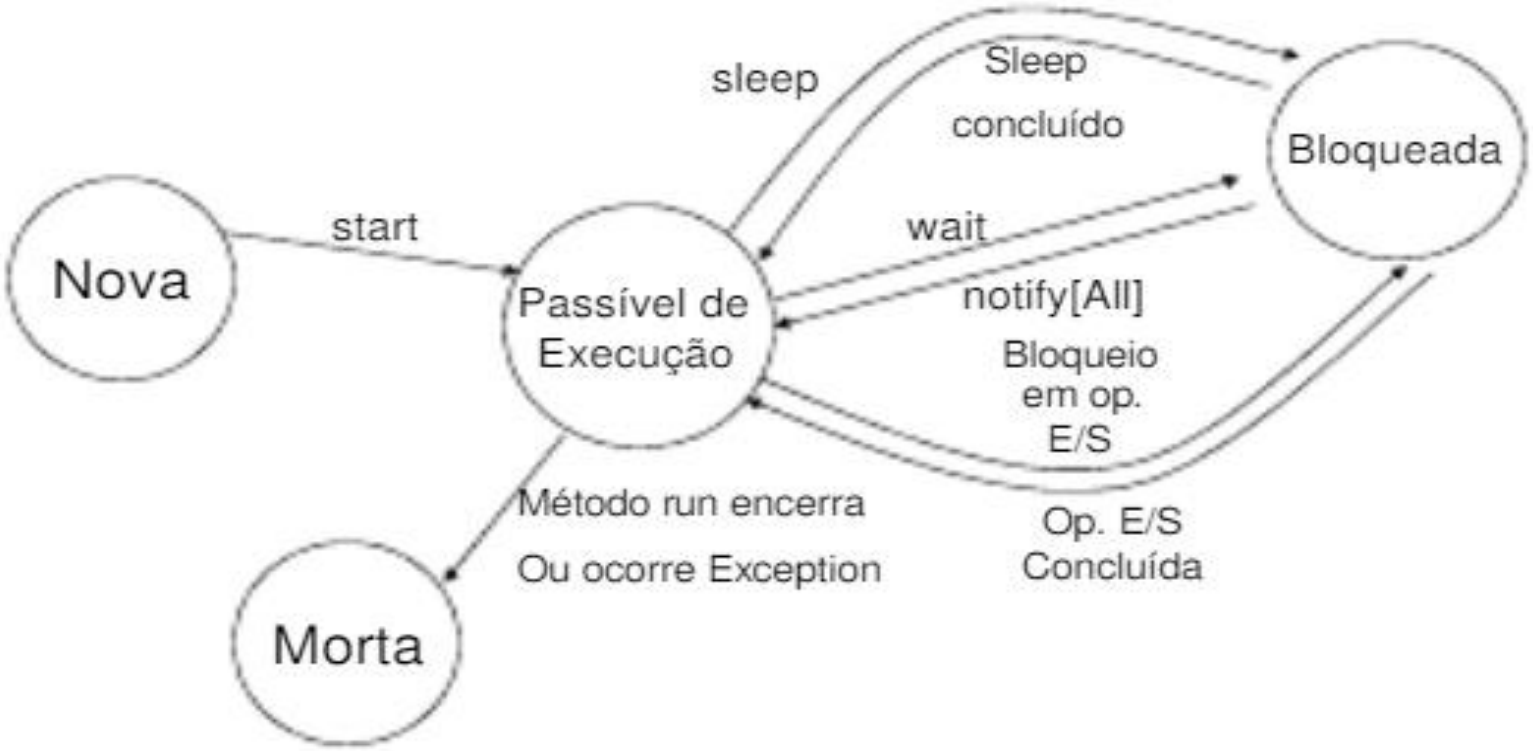

Nova
start
Passível de Execução
sleep
Sleep
concluído
wait
notify[All]
Bloqueio em op. E/S
Op. E/S Concluída
Bloqueada
Método run encerra
Ou ocorre Exception
Morta

Estados

- Threads bloqueadas: Ocorre quando uma thread fica dormente (sleep) por um tempo pré-determinado, quando é colocada em espera (wait) ou quando a thread tenta bloquear (bloqueio de I/O) um objeto que já está bloqueado por outra thread.

- Threads mortas ou encerradas: Ocorre quando uma thread morre naturalmente (fim da execução do trecho de código incluído na thread) ou quando tem uma morte abrupta devido a ocorrência de alguma exceção não tratada no código.

Estados


Nova
start
Passível de Execução
sleep
Sleep
concluído
Bloqueada
wait
notify[All]
Bloqueio em op. E/S
Op. E/S Concluída
Método run encerra
Ou ocorre Exception
Morta

Prioridades

A prioridade de um thread corresponde a preferência que ela terá perante as demais durante sua execução. Quanto maior a prioridade de um thread, maior será sua preferência no uso da CPU. Threads de mesma prioridade costumam partilhar o tempo de CPU igualmente. A prioridade é extremamente ligada ao algoritmo de escalonamento de CPU que o sistema operacional utiliza. Para definir a prioridade de um thread, são usados números de 1 a 10 ou por classes, sendo que o número 5 é usado para definir a prioridade como normal. Quanto maior o valor da prioridade, maior é a sua prioridade.

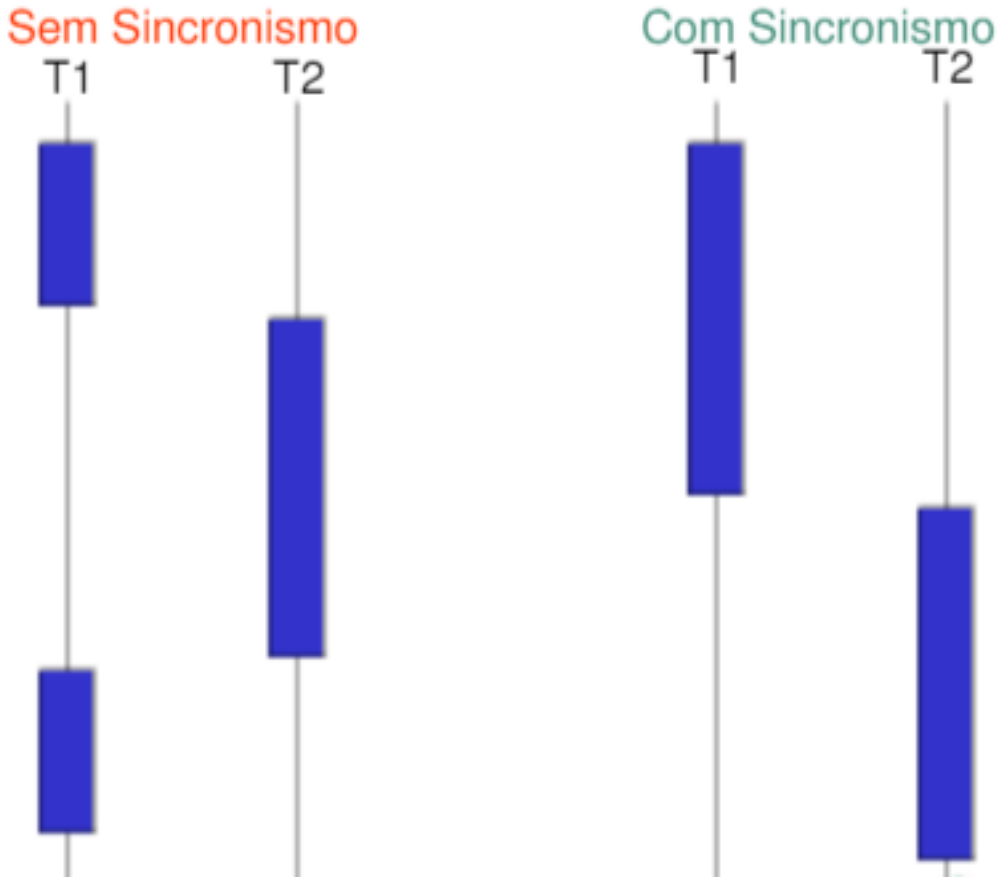
Prioridades

Entretanto, nem todos os sistemas operacionais possuem as mesmas prioridades de um thread Java. Portanto, para garantir que um thread com prioridade 10 tenha prioridade alta tanto em um sistema operacional cuja prioridade máxima seja 10 quanto em outro que seja 100, a JVM traduz a prioridade especificada no código para a do sistema operacional, logo uma prioridade 10 em Java pode ser traduzida para uma prioridade 100, por exemplo.

Sincronização

Durante a execução de threads, há casos em que elas trabalham independentemente uma da outra, sem necessidade de qualquer comunicação entre elas. Por outro lado, há casos em que elas se comunicam de alguma forma ou utilizam dados em comum. Este comportamento gera a necessidade de denominar os threads em assíncronas e síncronas, dependendo da forma de trabalho desempenhada.

Sincronização



Sincronização

Threads que trabalham independentes no tempo, são assíncronas enquanto aquelas que trocam informações em tempo de execução são síncronas. As threads se diferem dos processos por poderem ter áreas de dados comuns. Isto pode facilitar em muito a implementação de programas. Porém, pode causar alguns erros quando a mesma base de dados é alterada por mais de um thread, em momentos inesperados.

Sincronização

O uso de memória compartilhada entre os threads obriga o programador a sincronizar as ações de suas threads. Para isso, Java provê monitores ou locks. Imagine um lock como uma permissão para que apenas um thread possa utilizar um recurso por vez. Cada objeto em Java possui um lock e ele deve ser obtido através do comando synchronized.

Classe java.lang.Thread

- Serve para construir uma nova thread em qualquer ponto do seu código:
  - Instâncias ou classes derivadas poderão ter seus códigos processados em paralelo.

Classe java.lang.Thread

- Métodos principais:
  - **void run()**: método que deve ser sobrecarregado para incluir o código a ser processado em paralelo (não deve ser chamado diretamente);
  - **void start()**: prepara uma linha de execução para ser iniciada invocando o método run() posteriormente;
  - **static void sleep(long ms)**: coloca a thread corrente em estado de suspensão (dormente) por um período de tempo quantificado em milissegundos.

Exemplo

- Fazer um programa em Java que imprima na tela as strings: “Olá mundo” e “Threads!” 500 vezes cada uma, alternadamente.

- Transformar o exemplo anterior implementando threads.

Implementando Threads


```
public class Main
{
    public static void main(String[] args)
    {
        for(int i = 0; i < 500; i++)
        {
            System.out.println("Olá mundo!");
            System.out.println("Thread!");
        }
    }
}
```

Implementando Threads

- Podemos criar threads em java de duas formas:
  - Através da herança da classe Thread;
  - Através da implementação da interface Runnable.
- Nos dois casos a programação do processo é feita na implementação do método **run()**.
- A vantagem do uso da interface Runnable está em permitir que sua classe herde outra classe base e se comporte como uma thread ao mesmo tempo.

Implementando Threads


```
public class Mensagem implements Runnable
{
    private String mensagem;

    public Mensagem (String mensagem)
    {
        this.mensagem = mensagem;
    }

    @Override
    public void run() {
        // TODO Auto-generated method stub
        for (int i = 0; i < 500; i++)
            System.out.println(this.mensagem);
    }

}
```

Implementando Threads


```
public class Main
{
    public static void main(String[] args)
    {
        Mensagem msg1 = new Mensagem("Olá mundo!");
        Mensagem msg2 = new Mensagem("Threads!");

        Thread thread1 = new Thread(msg1);
        Thread thread2 = new Thread(msg2);

        thread1.start();
        thread2.start();
    }
}
```

OBRIGADO