Pontifícia Universidade Católica de Minas Gerais
Instituto de Ciências Exatas e Informática
Departamento de Ciência da Computação
Curso de Ciência da Computação

Laboratório de Redes e SO

Processos e Threads

Objetivos: Conhecer o conceito e aplicação de processos e threads

Processo.

Um processo no sistema operacional (SO) representa uma tarefa (ou um programa) em execução. Por exemplo, se você executa um programa que faz o envio de um texto pela rede, um processo será iniciado no SO e ele irá executar essa atividade. Alguns processos são “visíveis” pelos usuários, através de interfaces como um browser ou um editor de texto enquanto outros processos podem ficar em execução apenas, como um servidor de DHCP ou DNS.

Uma outra definição de processo poderia ser: “softwares que executam alguma ação e que podem ser controlados de alguma maneira, seja pelo usuário, pelo aplicativo correspondente ou pelo sistema operacional.”

A estrutura básica de um processo é formada por:

- Uma imagem do código executável (software) associado;
- A memória que contém o código executável e dados específicos da execução do processo;
- A descrição de recursos do sistema alocados ao processo;
- Informações de atributos de segurança
- Indicação do estado atual do processo.

Ciclo de Vida do Processo

Um processo passa por diferentes estados desde sua criação até seu término. O sistema operacional reúne todas essas informações através de estruturas específicas chamadas PCB (sigla de Process Control Blocks, o que em tradução livre seria Blocos de Controle de Processos).

• Enquanto ele é criado, seu estado é considerado "Novo";

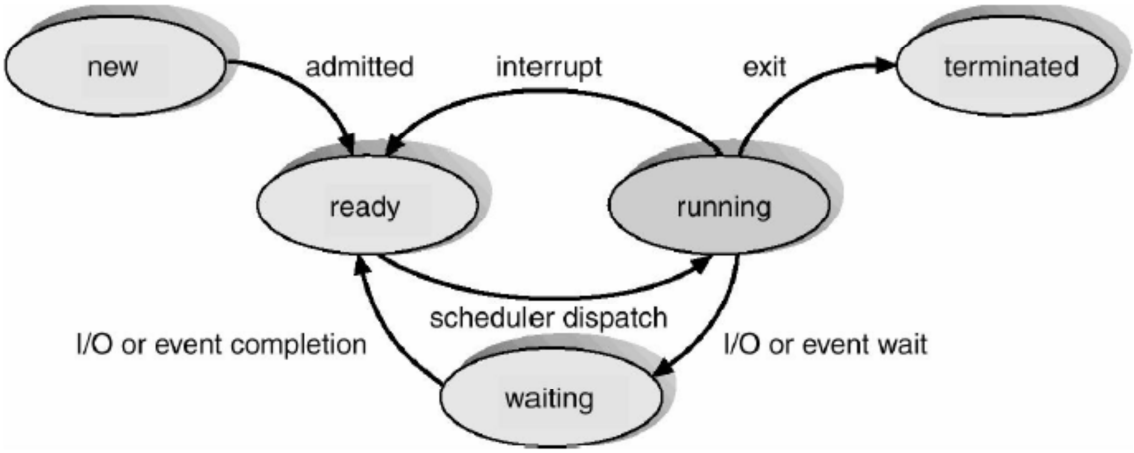
• Quando pode ser escalonado para rodar fica em “Pronto”.

• Em ação, muda para "Executando";

• Quando depende da ocorrência de algum evento, vira "Esperando";

• Quando não mais necessário, o processo é "Terminado".

Veja o Diagrama a seguir:



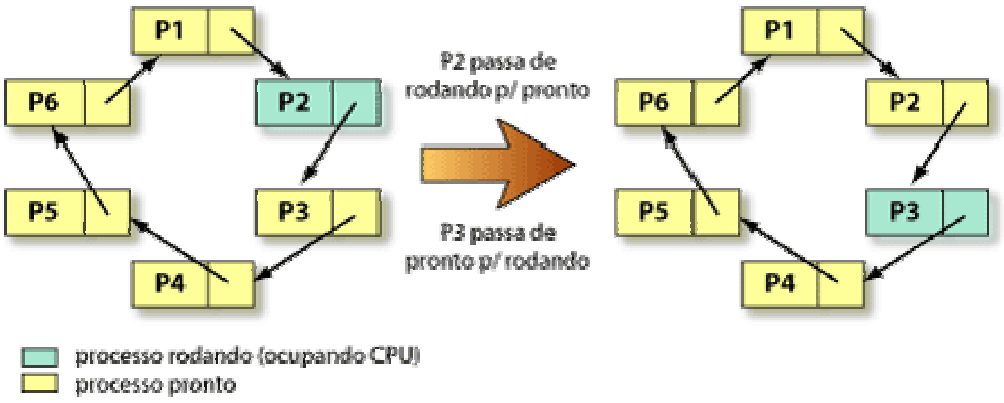
Escalonamento de Processos

Nem todos os processos existentes estão executando o tempo todo. Como existem mais processo do que processadores, de fato o SO executa apenas alguns poucos processos (numero de núcleos) a cada momento. O SO possui um sistema de escalonamento que define qual processo vai ser executado a cada momento. Desta forma ele está sempre trocando qual processo deve ser executado. Como isso é feito em intervalos de tempo muito pequenos temos a sensação de ter todos os processos em execução ao mesmo tempo.

Existem vários algoritmos de escalonamento de processos, como:

- FIFO (First in, first out) : o primeiro que chega será o primeiro a ser executado;
- SRT (Shortest Remaining Time): Neste algoritmo é escolhido o processo que possua o menor tempo restante, mesmo que esse processo chegue à metade de uma operação, se o processo novo for menor ele será executado primeiro;
- RR (Round-Robin): Nesse escalonamento o sistema operacional possui um timer, chamado de quantum, onde todos os processos ganham o mesmo valor de quantum para rodarem na CPU. Com exceção do algoritmo RR e escalonamento garantido, todos os outros sofrem do problema de Inanição (starvation).

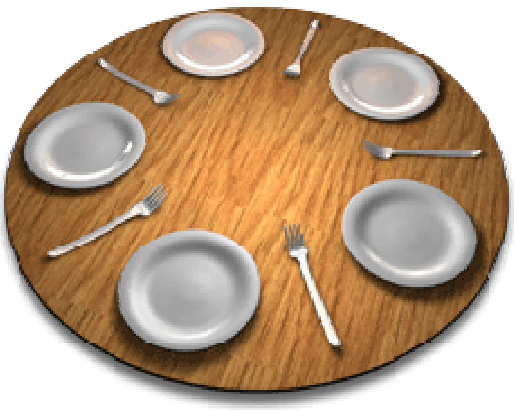
A figura a seguir mostra um exemplo de escalonamento de processos por Round-Robin.



Concorrência e Hierarquia de Processos.

Quando temos vários processos em execução, temos concorrência, pois existem vários processos disputando o acesso a recursos partilhados, como a CPU. A gestão da concorrência entre processos é a fonte de inúmeras dificuldades no desenvolvimento de software. O acesso não coordenado a um recurso pode induzir a um comportamento imprevisível ou a um travamento (dead-lock).

Um problema clássico de concorrência é conhecido como o jantar dos filósofos.



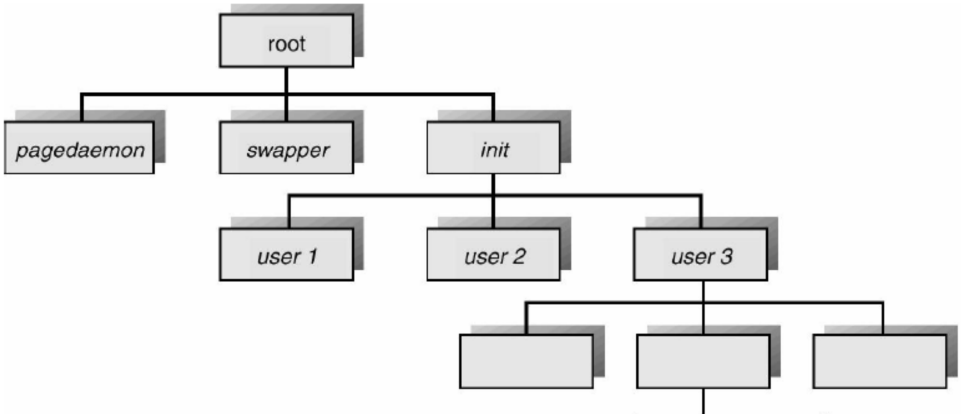
N filósofos estão sentados ao redor de uma mesa redonda tendo um prato de macarrão diante de si. Entre cada dois pratos existe um garfo e a vida de um filósofo é pensar, comer, pensar, comer e assim indefinidamente.

No entanto, para comer, o filósofo precisa necessariamente de dois garfos (um que está à sua esquerda e o outro à sua direita).

Quando terminou de pensar (por ter fome), tenta pegar os garfos vizinhos para poder comer. Ao terminar de comer, devolve os garfos à mesa.

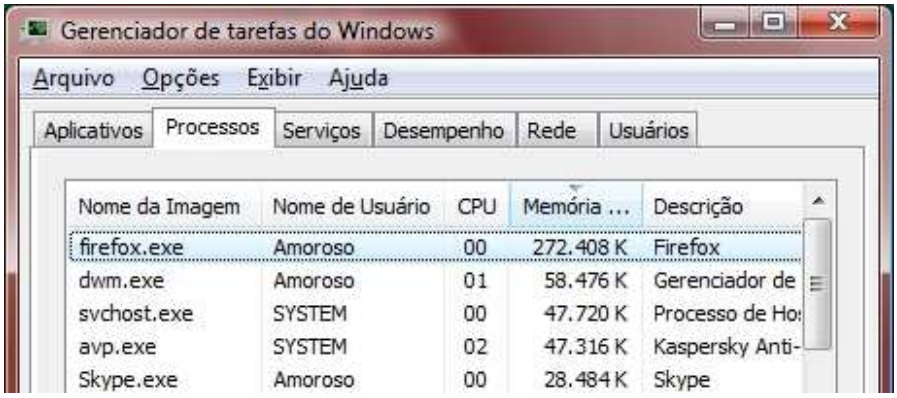
Neste problema, se a ordem em que os filósofos pegam o garfo não for bem definida, podemos ter um deadlook.

Os processos também possuem uma hierarquia de criação. Um processo pode iniciar outros processos. A figura a seguir apresenta um exemplo de hierarquia de processos no UNIX.



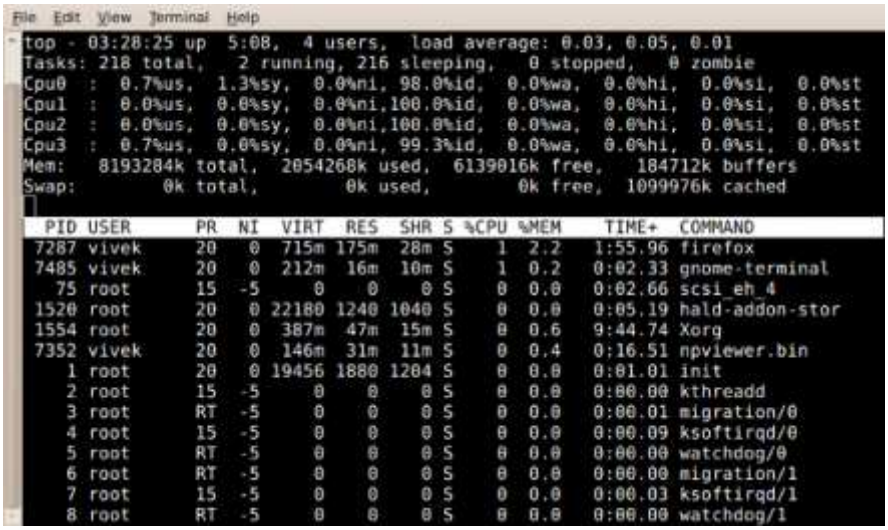
Listar Processos no SO.

No Windows, podemos usar o Gerenciador de Tarefas para visualizar os processos em execução. Para acessá-lo: use a combinação Ctrl+Alt+Del e selecione Gerenciador de Tarefas. Com a janela aberta, acesse a aba “Processos”.



Nesta interface, além de ver quais processos estão em execução, podemos ver o consumo de CPU e Memória de cada um deles.

No Linux existe um comando chamado “*top”*. Este comento lista os processos em execução, alem de mostrar algumas informações sobre os recursos em uso do servidor.

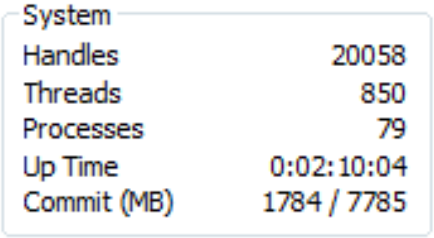


Existem também o comando *pstree*. Este comando mostra os processos e a hierarquia entre eles.

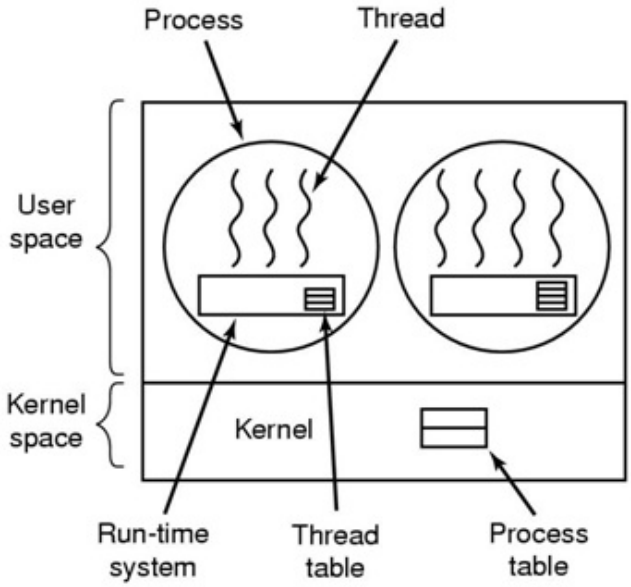
Threads

Thread (ou processo leve) é um fluxo de execução dentro do mesmo processo compartilhando dados e recursos do processo. É falado processo leve dado o menor tempo gasto em atividades de criação e escalonamento da thread, se comparadas aos processos. O compartilhamento de memória entre as threads maximiza o uso dos espaços de endereçamento e torna mais eficiente o uso destes dispositivos

A thread é uma forma de um processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas concorrentemente. O suporte a thread é fornecido pelo próprio SO ou implementada através de uma biblioteca de uma determinada linguagem. Em geral cada processo “dispara” várias threads. Na figura a seguir, do gerenciador de tarefas do Windows, pode ser visto o número de processo e threads ativas.



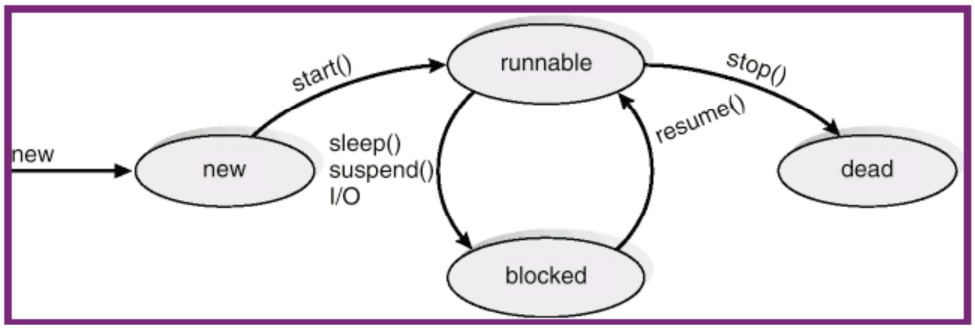
A figura a seguir ilustra como as threads operam. Um processo que é controlado pelo SO pode criar e gerenciar várias threads. Cada uma tem uma linha de execução independente.



Ciclo de Vida da Thread

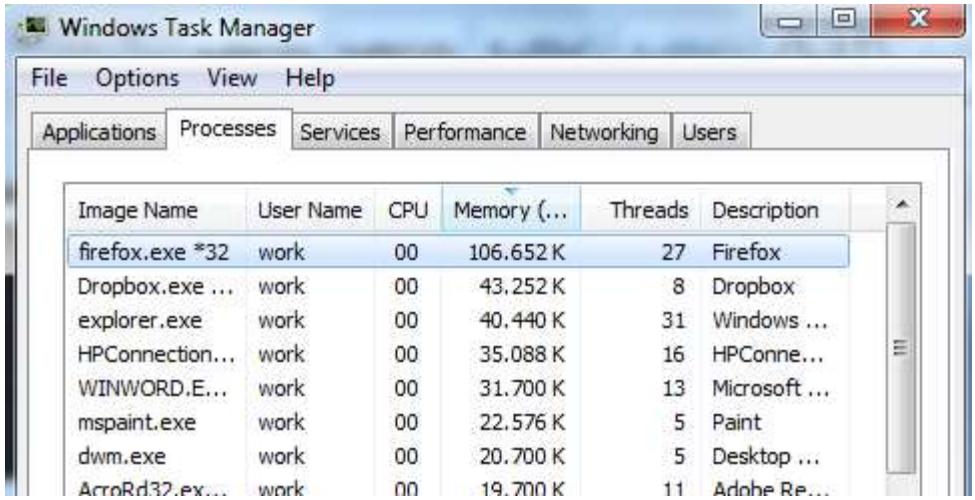
Assim como os processos, as threads tem um “ciclo de vida” e podem assumir os seguintes estados:

- Nova: logo após ser criada (antes do start);
- Executando: após ser ativada (start)
- Bloqueda: ao ter sua execução interrompida;
- Morta: após ser parada.



Identificar Threads no Windows

No Windows, podemos usar o Gerenciador de Tarefas para visualizar quantas threads cada processo está executando. Vá no menu “View” em “Select Columns” e marque a opção “Threads”



Threads em Java

A classe Thread é uma classe nativa da linguagem Java, que permite representar um fluxo independente de execução dentro de um programa. Uma subclasse de Thread exige a implementação do método *run()*, que contém o código a ser executado pela thread em si.

A seguir o exemplo de código de uma classe filha de Thread imprime números de 1 a 100 a cada um segundo.

```
public class escrita extends Thread {
     public void run() {
          for (int i=0; i<100; i++) {
               System.out.println("Número :" + i);
               try {
                    Thread.sleep(1000);
               } catch (InterruptedException e) {
                    e.printStackTrace();
               }
          }
     }
}
```

Para criar uma thread, propriamente dita, a partir de uma subclasse de Thread, basta declarar um objeto desta classe, utilizando o operador new para instanciá-la. Instanciado o objeto a thread está pronta para ser rodada. Para isto, deve-se chamar o método start(), responsável por providenciar o seu escalonamento pelo sistema operacional e por executar o método run().

```
public class principal {
     public static void main(String[] args) {
          //Cria o contexto de execução
          escrita eThread = new escrita();
          //Ativa a thread
          eThread.start();
     }
}
```

Atividades:

1. Para o programa dado no texto, executar o mesmo e verificar no Gerenciador de tarefas do Windows quando de memória o processo está consumindo e quantas threads estão associadas a ele. Dicar, o main pode ficar na mesma classe do método escrita.
2. Dado o programa pingpong a seguir, executar e fazer a mesma verificação do item 1.
3. Verificar como é a saída do programa pingpong e explicar o porquê da ordem dos textos de saída.

```
public class pingpong extends Thread {
     private String palavra;
     public pingpong(String palavra) {
          this.palavra= palavra;
     }

     public void run() {
          try {
               for(int i= 0; i < 30; i++) {
                    System.out.print("\n" + palavra + " "
                                          + " : " + i);
               if (palavra.equals("ping"))
                    Thread.sleep(200);
               else
                    Thread.sleep(100);
          }
     }
     catch (InterruptedException e) {
           return;
     }

     public static void main(String [] args) {
          Thread t1= new pingpong("ping");
          Thread t2= new pingpong("PONG");
          t1.start();
          t2.start();
     }
}
```

4. A partir do programa pingpong, gerar um programa tênis de duplas, com “4 jogadores” em execução simultaneamente. Cada jogador basicamente deve imprimir seu nome e a palavra REBATE. Em seguida, aguardar um segundo. Crie dois times e certifique-se que após um time rebater a bola apenas um jogador do outro time rebata, ou seja, se o jogador do time 1 bater na bola, apenas os jogadores do time 2 poderão rebater agora. Dica use região crítica com uma variável que armazena o último time que rebateu a bola!

```
public static synchronized void play(int team)
    {
        if (lastTeamPlay != team)
            lastTeamPlay = team;
    }
```

5. Acesse o site https://www.caelum.com.br/apostila-java-orientacao-objetos/apendice-sockets/index.html, leia as instruções, semelhantes da aula anterior e execute o código multithread para rede do item 19.11.