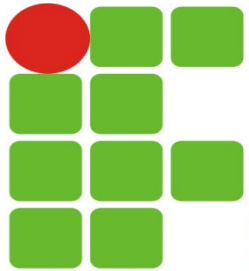


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SUL-RIO-GRANDENSE
Campus Passo Fundo

Ensino Médio Integrado
Técnico em Informática

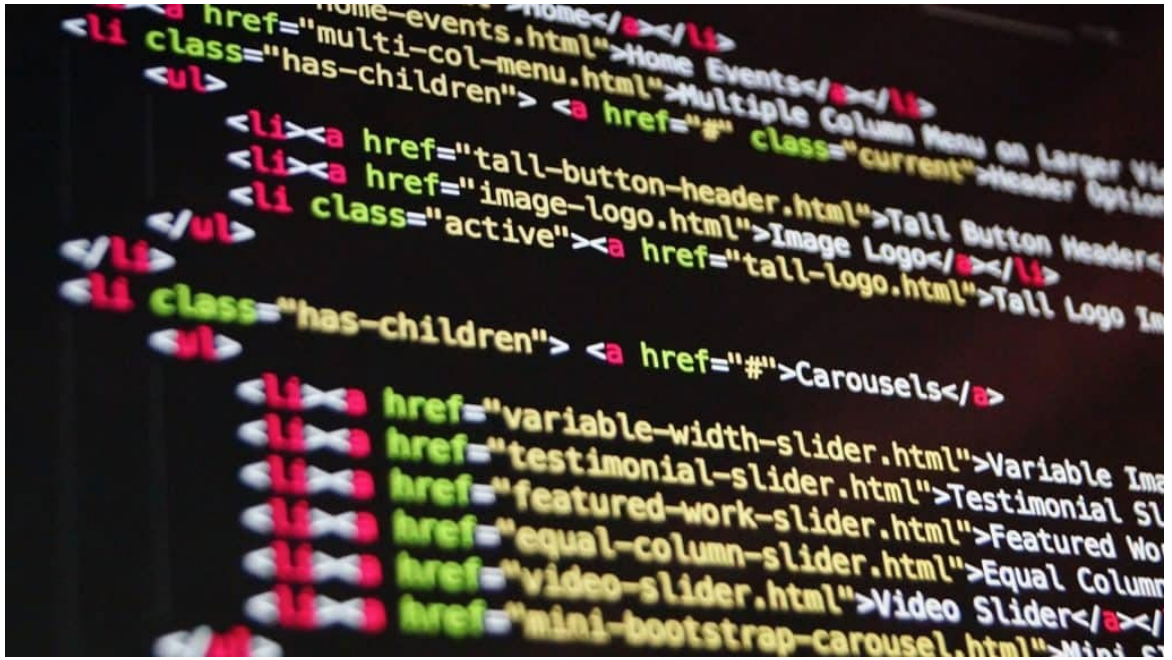
Sistemas Operacionais

Prof. Lisandro Lemos Machado
Profª. Ieda Rosana Kolling Wiest

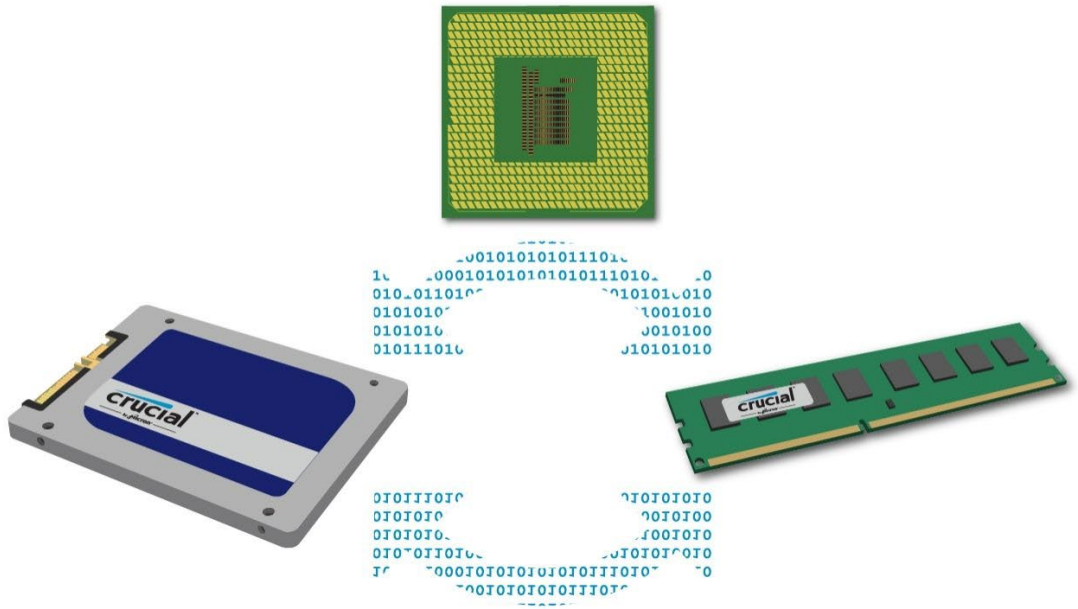
Gerências do Sistema Operacional

Programação

- Programa
  - Código passivo a ser executado;
  - Estático – guardado no HD;
  - Conjunto de instruções sequenciais;
  - Um programa pode ser executado por vários processos;
  - Cada execução com seus próprios dados;

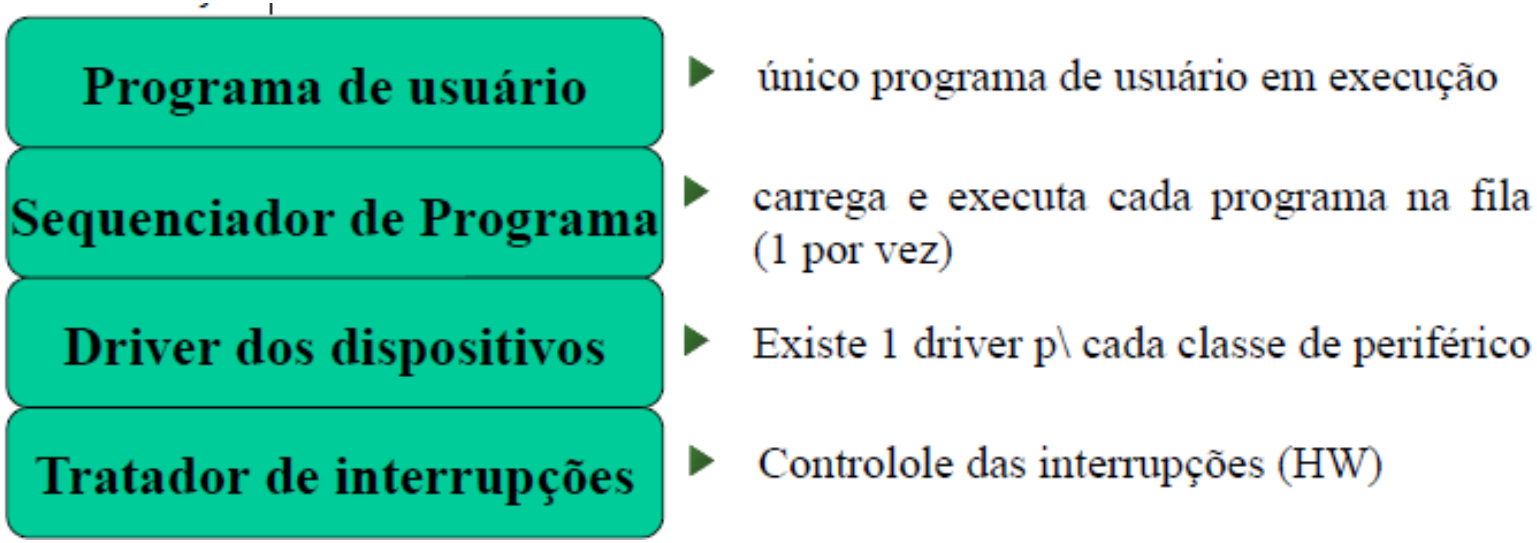


Processo



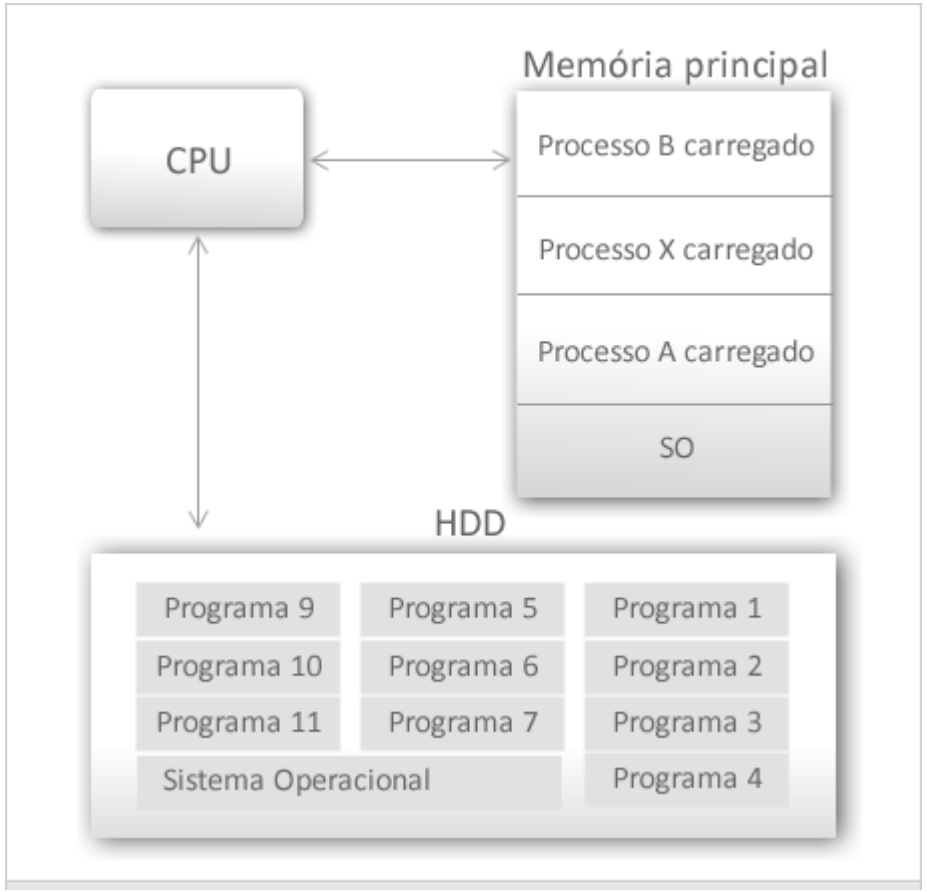
- Programa em execução na memória;
- Possui um ciclo de vida;
- É dinâmico, porque muda seu estado;
- Faz chamadas de sistema;
- Cria outros processos;
- Possui uma identidade única no sistema
  - PID (Process ID)
  - Duas execuções de um mesmo programa vão ser diferenciadas Pois são controladas por processos diferentes (PIDs diferentes)

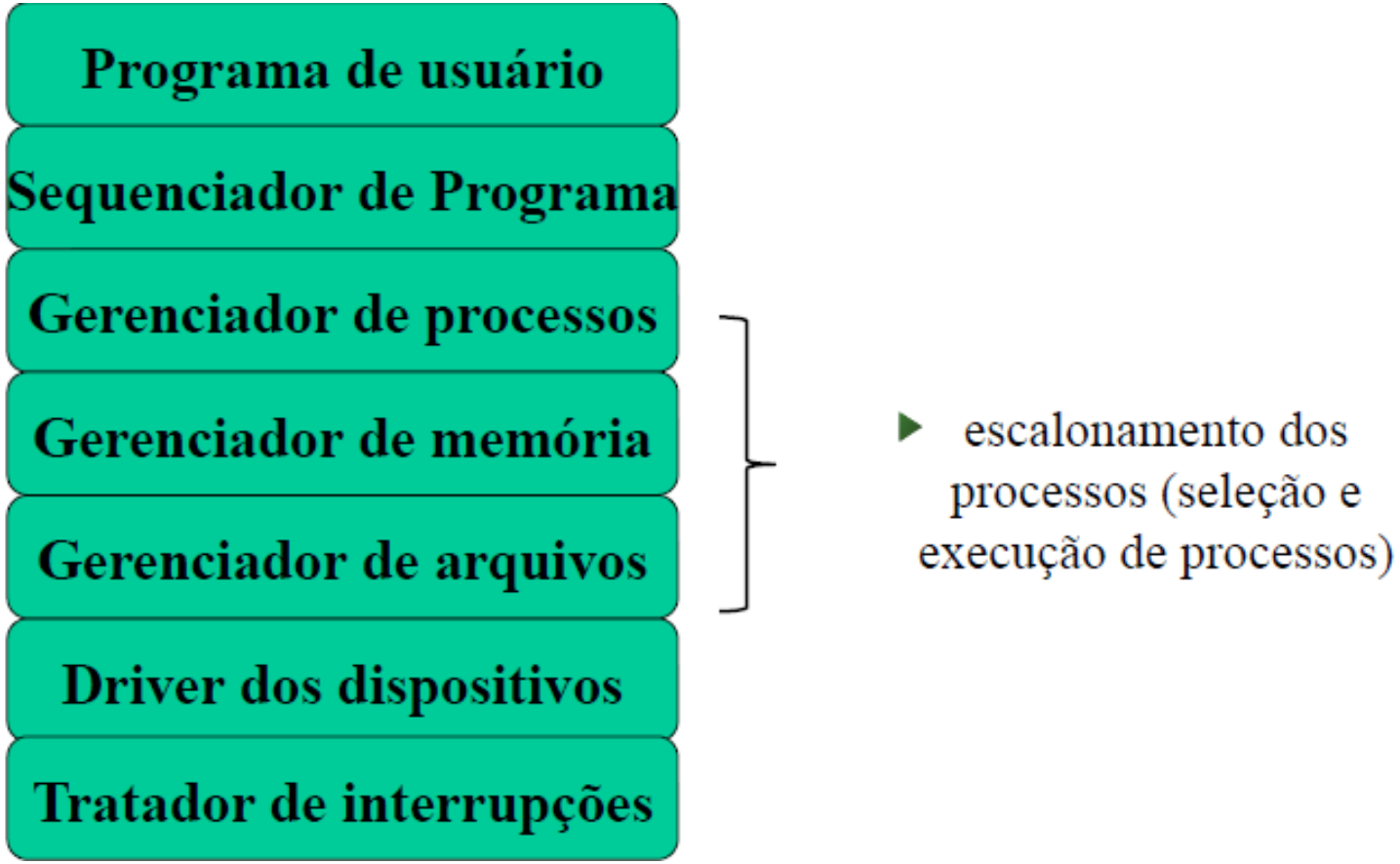
- Os sistemas operacionais podem ser classificados como monoprogramados ou multiprogramados;
  - **Monoprogramação**: Um único programa de usuário em execução



- **Multiprogramação**: vários programas de usuário em execução
- tempo é distribuído entre os diversos programas
- programa executa até:
  - acabar o seu tempo
  - acontecer uma entrada ou saída de dados (E/S)
  - espera por evento

- Dois elementos foram importantes para o surgimento da multiprogramação:
  - O uso das interrupções e o desenvolvimento dos discos magnéticos;





- escalonamento dos processos (seleção e execução de processos)

Serviços

- Do ponto de vista de projeto o sistema operacional é um gerenciador de recursos, tais como:

  - Processos:
    - Criação e deleção de processos;
    - Suspensão e retomada de processos;
    - Sincronização de processos;
    - Comunicação de processos;
    - Dependência entre processos (deadlocks);

- **Memória**
  - Uso das regiões de memória por difentes processos;
  - Estratégias de alocação dos espaços de memória;

- **Arquivos**
  - Criação e deleção de arquivos/diretórios
  - Mapeamento para memória secundária

- **Dispositivos de E/S**
  - Acesso, leitura e gravação de dados em dispositivos;

- Vida de um processador:
  - https://www.youtube.com/watch?v=DnGo1c6qPG0

Processos

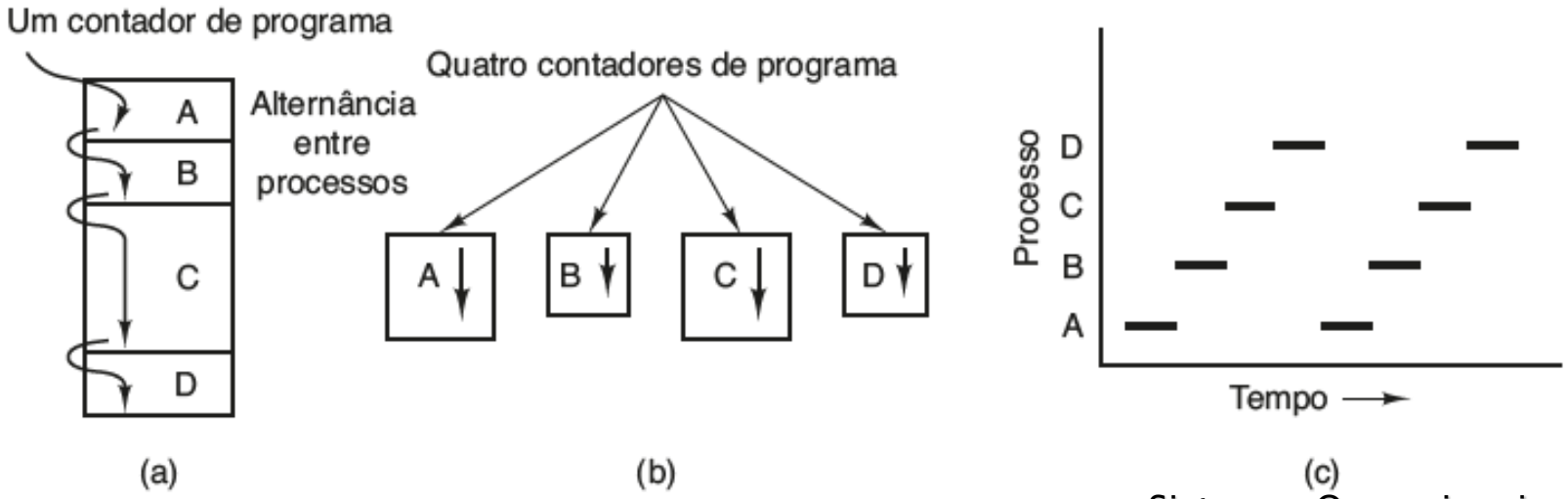
- Todos os softwares executáveis no computador são organizados em uma série de processos sequenciais
  - Incluindo o sistema operacional;

- Processo
  - É uma instância de um programa em execução;
  - Inclui os valores atuais do contador do programa, registradores e variáveis;
  - Cada processo tem sua própria CPU virtual.

Multiprogramação

- Mecanismo de trocas rápidas;
- CPU real troca a todo momento de processo em processo;
- Processos sendo executados em (pseudo) paralelismo.
- Ex: computador multiprogramado com 4 programas na memória:

(a) Multiprogramação de quatro programas. (b) Modelo conceitual de quatro processos sequenciais independentes. (c) Apenas um programa está ativo de cada vez.



- Cada processo com seu próprio fluxo de controle e sendo executado independente dos outros (seu próprio contador de programa lógico);
- Quando cada processo é executado, o seu contador de programa lógico é carregado para o contador de programa real;
- Durante um intervalo longo o suficiente, todos os processos tiveram progresso, mas a qualquer dado instante apenas um está sendo de fato executado;
- Um processo é uma atividade de algum tipo. Ela tem um programa, uma entrada, uma saída e um estado.

Criação de Processos

- Quatro eventos principais fazem com que os processos sejam criados:
  - 1. Inicialização do sistema
    - 1º plano: interação com usuário. (Ex; ambiente gráfico)
    - 2º plano: funções específicas
  - 2. Execução de uma chamada de sistema de criação de processo por um processo em execução
    - criar um ou mais processos novos para ajudá-lo
  - 3. Solicitação de um usuário para criar um novo processo
    - Digitando um comando, clicando sobre um ícone ou com a voz
  - 4. Início de uma tarefa em lote
    - Sequência de processos em lote

Término de Processos

- Cedo ou tarde, o novo processo terminará, normalmente devido a uma das condições:
  - 1. Saída normal (voluntária)
    - Finalizou seu trabalho
  - 2. Erro fatal (involuntário ➔ SO finaliza)
    - Erro causado pelo processo
  - 3. Saída por erro (voluntária)
    - Processo descobre erro fatal
  - 4. Morto por outro processo (involuntário ➔ usuário finaliza)
    - Chamada de sistema dizendo ao sistema operacional para matar outro processo

Estados de Processos

- Os três estados nos quais um processo pode se encontrar:
  - 1. Em execução (realmente usando a CPU naquele instante).
  - 2. Pronto (executável, temporariamente parado para deixar outro processo ser executado).
  - 3. Bloqueado (incapaz de ser executado até que algum evento externo aconteça).

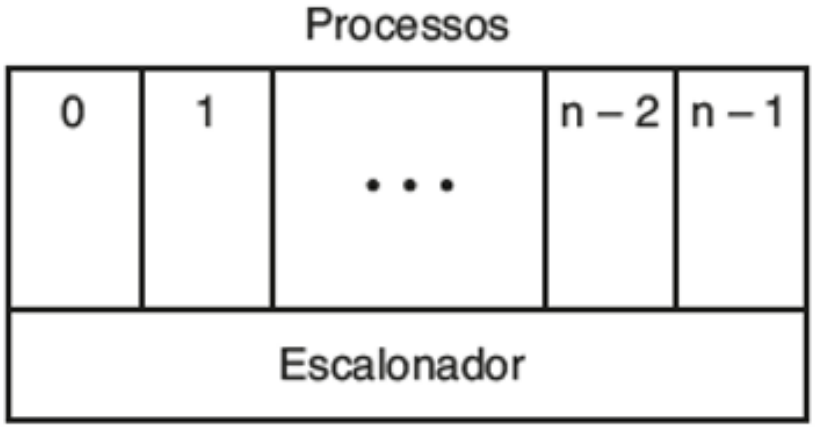
Um processo pode estar nos estados em execução, bloqueado ou pronto. Transições entre esses estados ocorrem como mostrado.

1. O processo é bloqueado aguardando uma entrada
2. O escalonador seleciona outro processo
3. O escalonador seleciona esse processo
4. A entrada torna-se disponível

- O escalonamento irá decidir qual processo deve ser executado, quando e por quanto tempo

- O nível mais baixo do sistema operacional é o escalonador, com uma variedade de processos acima dele.
  - Todo o tratamento de interrupções e detalhes sobre o início e parada de processos estão ocultos nele
    - O resto do sistema operacional é bem estruturado na forma de processos

O nível mais baixo de um sistema operacional estruturado em processos controla interrupções e escalonamento. Acima desse nível estão processos sequenciais.



Implementação de Processos

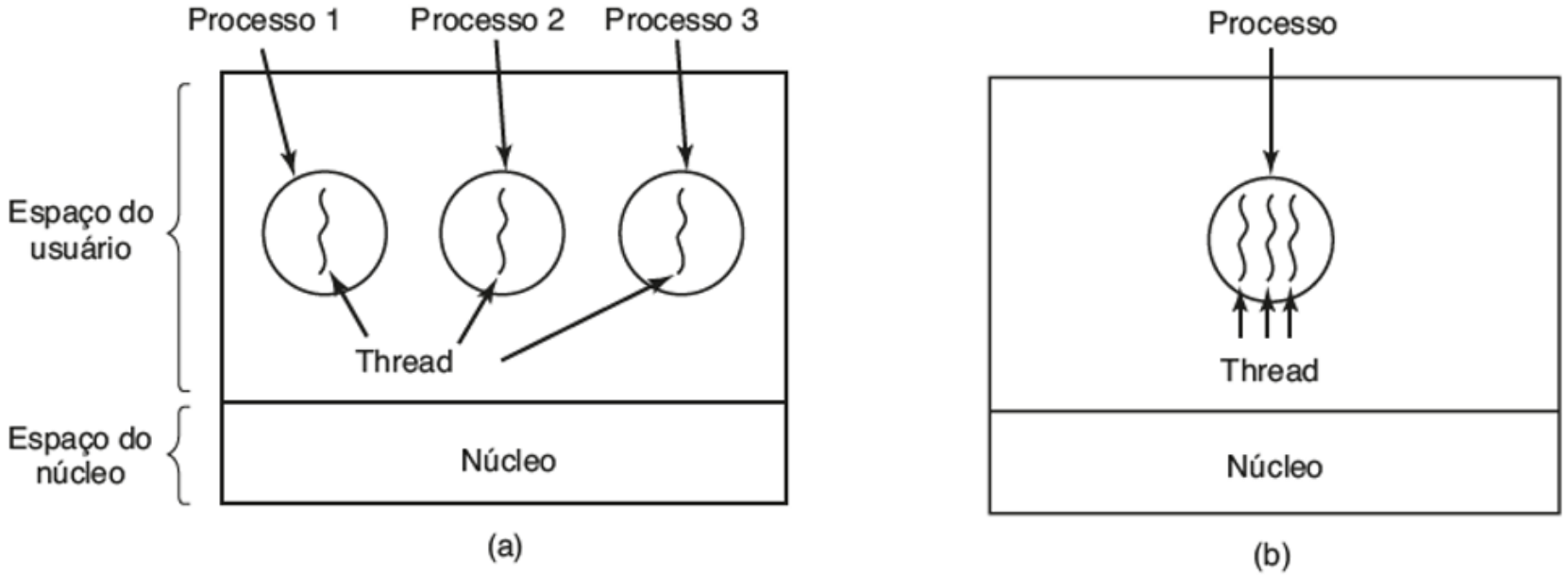
- Para o modelo de processos, o sistema operacional mantém uma tabela (um arranjo de estruturas)
  - Tabela de processos, com uma entrada para cada um deles.
  - Contêm informações importantes sobre o estado do processo:
    - Seu identificador (ID)
    - Seu contador de programa
    - Ponteiro de pilha
    - Alocação de memória
    - Estado dos arquivos abertos
    - Informação sobre sua contabilidade e escalonamento e tudo o mais que deva ser salvo quando o processo é trocado do estado em execução para pronto ou bloqueado, de maneira que ele possa ser reiniciado mais tarde como se nunca tivesse sido parado.

- Um processo pode ser interrompido milhares de vezes durante sua execução, mas a ideia fundamental é que, após cada interrupção, o processo retorne precisamente para o mesmo estado em que se encontrava antes de ser interrompido.
  - Estado do processamento é salvo a cada vez que ele passa da “execução” para “pronto” ou bloqueado

Modelo de Threads

- Processo é um modo para agrupar recursos relacionados
  - Tem um espaço de endereçamento contendo o código e os dados do programa, assim como outros recursos.
- Um processo possui, ao menos, uma linha (*thread*) de execução ➔ **thread**.
  - O thread possui:
    - Um contador de programa que controla qual instrução deve ser executada em seguida;
    - Ele tem registradores, que armazenam suas variáveis de trabalho atuais;
    - Tem uma pilha, que contém o histórico de execução, com uma estrutura para cada rotina chamada, mas ainda não retornada.
    - Threads acrescentam para o modelo de processo que ocorram múltiplas execuções no mesmo ambiente, com um alto grau de independência uma da outra

(a) Três processos, cada um com um thread. (b) Um processo com três threads.



- Na Figura (a) existem 3 processos tradicionais.
  - Cada processo tem seu próprio espaço de endereçamento e um único thread de controle.
- Na Figura (b) existe 1 único processo com 3 threads de controle.
  - Os três threads compartilham o mesmo espaço de endereçamento;
  - Podem acessar todo espaço de endereçamento de memória dentro do espaço de endereçamento do processo;
    - Pode ler, escrever, ou mesmo apagar a pilha de outro thread;
  - Não há proteção entre threads
    - Múltiplos threads de maneira que eles possam cooperar, não lutar

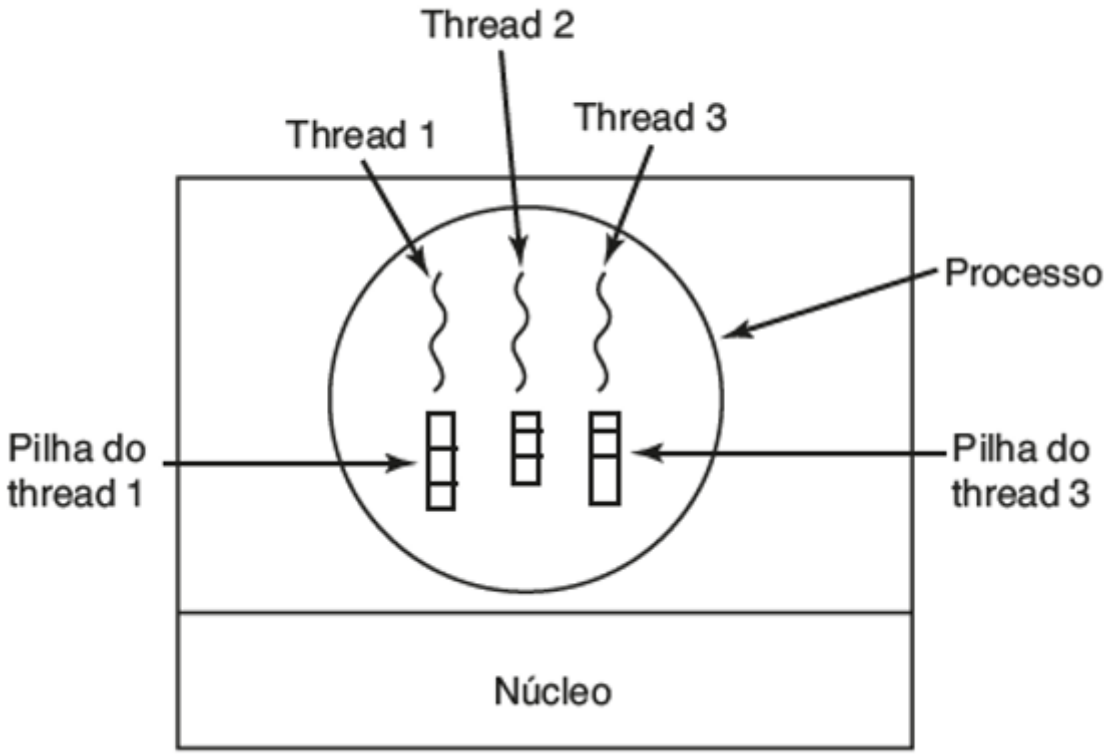
A primeira coluna lista alguns itens compartilhados por todos os threads em um processo. A segunda lista alguns itens específicos a cada thread.

Itens por processo	Itens por thread
Espaço de endereçamento	Contador de programa
Variáveis globais	Registradores
Arquivos abertos	Pilha
Processos filhos	Estado
Alarmes pendentes	
Sinais e tratadores de sinais	
Informação de contabilidade	

- Capacidade para múltiplos threads de execução de compartilhar um conjunto de recursos de maneira que possam trabalhar juntos intimamente para desempenhar alguma tarefa;
- Um thread pode estar em qualquer um de vários estados: em execução, bloqueado, pronto, ou concluído;
- Cada pilha do thread contém uma estrutura para cada rotina chamada, mas ainda não retornada
  - Essa estrutura contém as variáveis locais da rotina e o endereço de retorno para usar quando a chamada de rotina for encerrada.

- Threads podem ser implementados no espaço do usuário ou no núcleo.

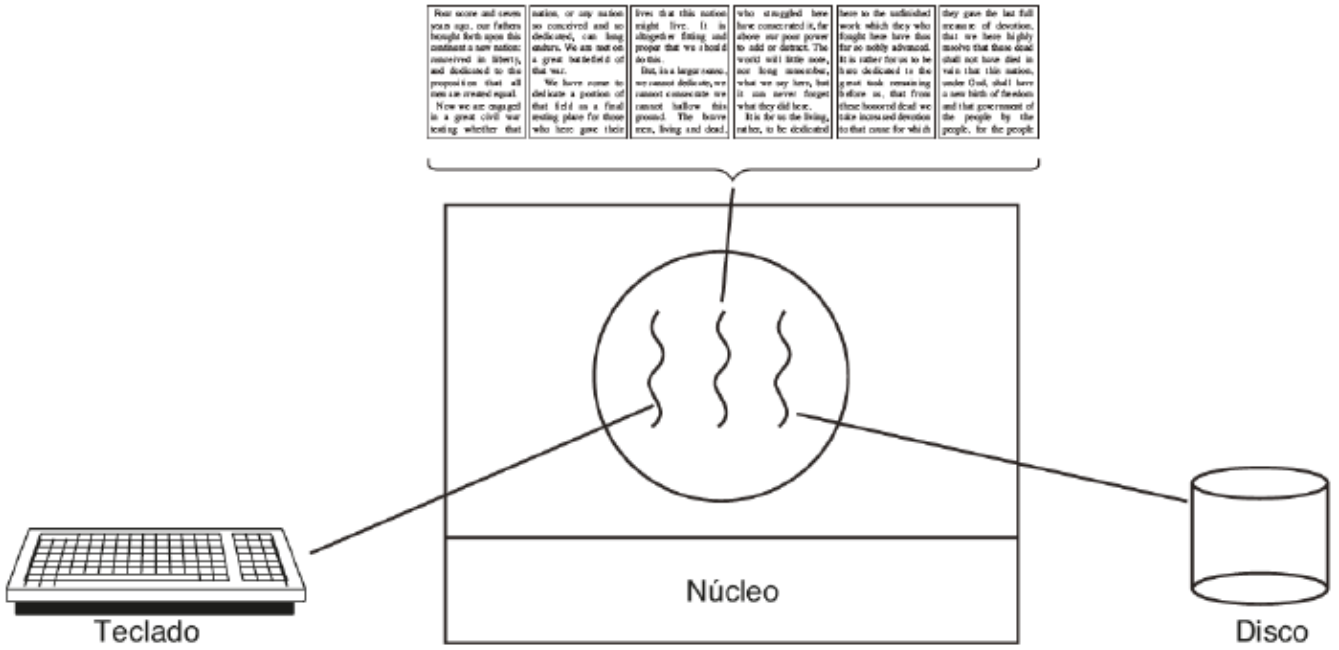
Cada thread tem a sua própria pilha.



Threads

- Quando há necessidade de um processo se dividir em partes menores para agilizar sua(s) atividade(s);
- Ex: processador de texto
  - Um thread interage com o usuário e o outro lida com a reformatação em segundo plano;
  - Uma thread realiza formatação solicitada por usuário enquanto thread interativa segue respondendo ações de mouse e teclado do usuário;
  - Capacidade de salvar automaticamente o arquivo inteiro para o disco em intervalos de poucos minutos ➔ terceiro thread pode fazer backups de disco sem interferir nos outros dois.

Um processador de texto com três threads



- Os três threads precisam operar no documento.
  - Ao existirem três threads em vez de três processos, eles compartilham de uma memória comum e desse modo têm acesso ao documento que está sendo editado;
  - Com três processos isso seria impossível.
- São úteis em sistemas com múltiplas CPUs, onde o paralelismo real é possível

Comunicação entre processos

- Processos podem comunicar-se uns com os outros usando primitivas de comunicação entre processos, por exemplo, semáforos, monitores ou mensagens.
  - Um processo pode estar sendo executado, ser executável, ou bloqueado, e pode mudar de estado quando ele ou outro executar uma das primitivas de comunicação entre processos. A comunicação entre threads é similar.
  - Essas primitivas são usadas para assegurar que jamais dois processos estejam em suas regiões críticas ao mesmo tempo, uma situação que leva ao caos.

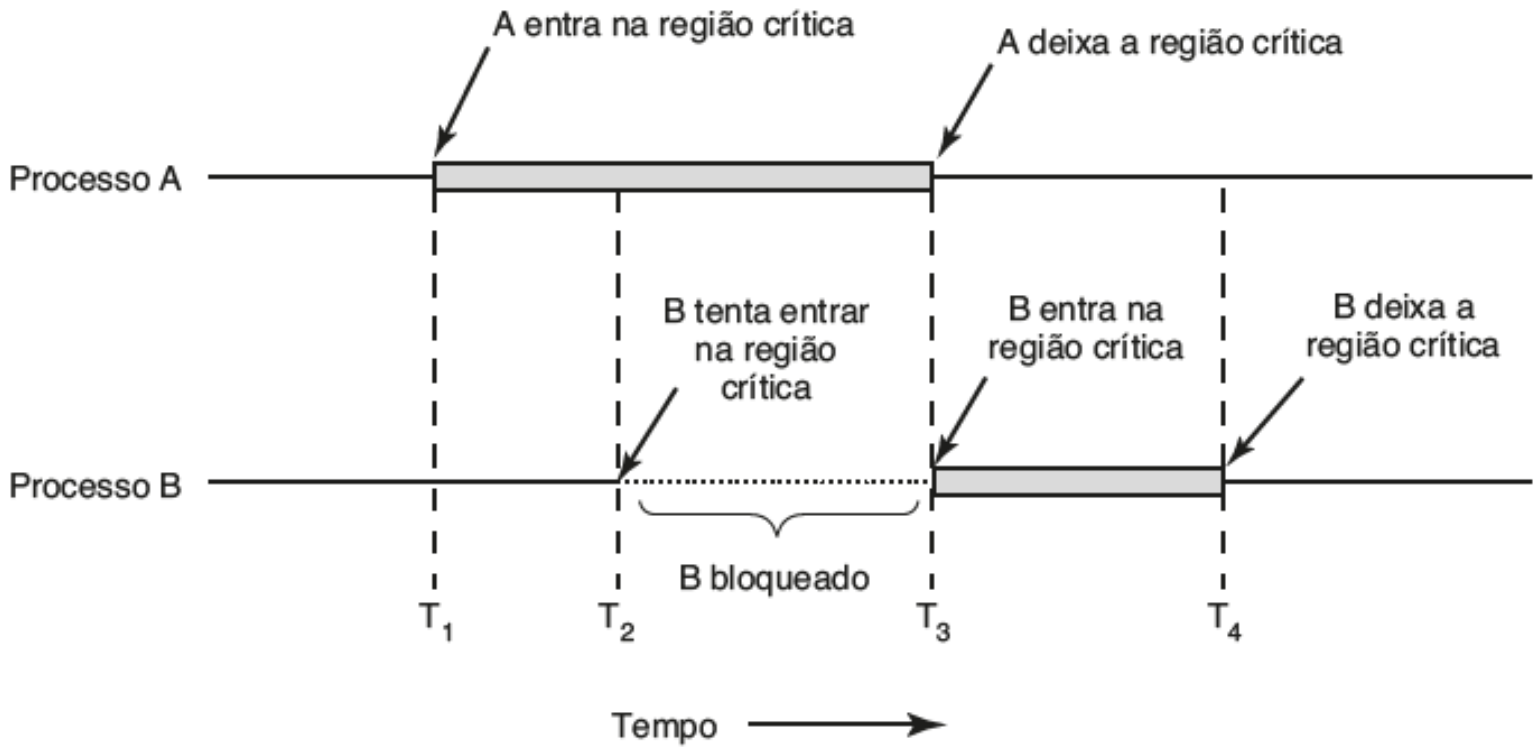
- Situações
  - Como um processo pode passar informações para outro;
  - Certificar-se de que dois ou mais processos não se atrapalhem;
  - Sequenciamento adequado quando dependências estão presentes.

Regiões críticas

- **Região crítica** ou **seção crítica**
  - Às vezes um processo tem de acessar uma memória ou arquivos compartilhados, ou realizar outras tarefas críticas que podem levar a corridas e disputas ➔ **conflito**!
  - Chave para evitar problemas em situações envolvendo memória compartilhada, arquivos compartilhados e tudo o mais compartilhado ➔ encontrar alguma maneira de proibir mais de um processo de ler e escrever os dados compartilhados ao mesmo tempo;

- Solução ➔ **exclusão mútua**
  - Se certificar de que se um processo está usando um arquivo ou variável compartilhados, os outros serão impedidos de realizar a mesma coisa.

- Precisamos que quatro condições se mantenham para chegar a uma boa solução:
  - 1. Dois processos jamais podem estar simultaneamente dentro de suas regiões críticas;
  - 2. Nenhuma suposição pode ser feita a respeito de velocidades ou do número de CPUs;
  - 3. Nenhum processo executando fora de sua região crítica pode bloquear qualquer processo;
  - 4. Nenhum processo deve ser obrigado a esperar eternamente para entrar em sua região crítica;


A entra na região crítica
A deixa a região crítica
Processo A
B tenta entrar na região crítica
B entra na região crítica
B deixa a região crítica
Processo B
B bloqueado
T1
T2
T3
T4
Tempo

Escalonamento

- Quando um computador é multiprogramado, ele frequentemente tem múltiplos processos ou threads competindo pela CPU ao mesmo tempo
  - Essa situação ocorre sempre que dois ou mais deles estão simultaneamente no estado pronto.
- Se apenas uma CPU está disponível, uma escolha precisa ser feita sobre qual processo será executado em seguida.
- A parte do sistema operacional que faz a escolha é chamada de escalonador, e o algoritmo que ele usa é chamado de **algoritmo de escalonamento**.

- Computadores tornaram-se tão rápidos que a CPU dificilmente ainda é um recurso escasso
  - A maioria dos programas para computadores pessoais é limitada pela taxa na qual o usuário pode apresentar a entrada (digitando ou clicando), não pela taxa na qual a CPU pode processá-la;
  - Há aplicações que demandam mais da CPU, como reproduzir uma hora de vídeo de alta resolução enquanto se ajustam as cores;
- Em servidores de rede, a situação muda consideravelmente
  - Múltiplos processos muitas vezes competem pela CPU, de maneira que o escalonamento importa.

- Dispositivos móveis e nós em redes de sensores
  - Além disso, a duração da bateria é uma das restrições mais importantes nesses dispositivos, alguns escalonadores tentam otimizar o consumo de energia.
- Além de escolher o processo certo a ser executado, o escalonador também tem de se preocupar em fazer um uso eficiente da CPU
  - Uma troca do modo usuário para o modo núcleo

Quando escalonar

- Quando um novo processo é criado, uma decisão precisa ser tomada a respeito de qual processo, o pai ou o filho, deve ser executado;
- Uma decisão de escalonamento precisa ser tomada ao término de um processo;
  - Esse processo não pode mais executar (já que ele não existe mais), então algum outro precisa ser escolhido do conjunto de processos prontos.
- Quando um processo bloqueia para E/S, em um semáforo, ou por alguma outra razão, outro processo precisa ser selecionado para executar.

- Algoritmos de escalonamento podem ser divididos em duas categorias em relação a como lidar com interrupções de relógio.
  - **Não preemptivo**
    - Escolhe um processo para ser executado e então o deixa ser executado até que ele seja bloqueado (seja em E/S ou esperando por outro processo), ou libera voluntariamente a CPU;
    - Mesmo que ele execute por muitas horas, não será suspenso forçosamente.
  - **Preemptivo**
    - Escolhe um processo e o deixa executar por no máximo um certo tempo fixado;
    - Se ele ainda estiver executando ao fim do intervalo de tempo, ele é suspenso e o escalonador escolhe outro processo disponível para executar;
    - Este escalonamento exige que uma interrupção de relógio ocorra ao fim do intervalo para devolver o controle da CPU de volta para o escalonador.

Objetivos do algoritmo de escalonamento

- Em qualquer circunstância, a justiça é importante;
- Processos comparáveis devem receber serviços comparáveis;
  - Conceder a um processo muito mais tempo de CPU do que para um processo equivalente não é justo.
- Categorias diferentes de processos podem ser tratadas diferentemente;
- Outra meta geral é manter todas as partes do sistema ocupadas quando possível.
  - Se a CPU e todos os outros dispositivos de E/S podem ser mantidos executando o tempo inteiro, mais trabalho é realizado por segundo do que se alguns dos componentes estivessem ociosos.

Algumas metas do algoritmo de escalonamento sob diferentes circunstâncias.

Todos os sistemas

- Justiça — dar a cada processo uma porção justa da CPU
- Aplicação da política — verificar se a política estabelecida é cumprida
- Equilíbrio — manter ocupadas todas as partes do sistema

Sistemas em lote

- Vazão (*throughput*) — maximizar o número de tarefas por hora
- Tempo de retorno — minimizar o tempo entre a submissão e o término
- Utilização de CPU — manter a CPU ocupada o tempo todo

Sistemas interativos

- Tempo de resposta — responder rapidamente às requisições
- Proporcionalidade — satisfazer às expectativas dos usuários

Sistemas de tempo real

- Cumprimento dos prazos — evitar a perda de dados
- Previsibilidade — evitar a degradação da qualidade em sistemas multimídia

Referências

FERREIRA, Rubem E. Linux: Guia do Administrador do Sistema.
São Paulo: Novatec, 2003.

Materiais de aula de João Brezolin e Gabriel Santin;

OLIVEIRA, Rômulo Silva de; CARISSIMI, Alexandre da S.;

TOSCANI, Simão Sirineo. Sistemas operacionais. 3. Ed.Porto Alegre: Bookman; UFRGS, 2008.

TANENBAUM, Andrew S. Sistemas operacionais modernos. 2.ed.
São Paulo: Pearson Prentice Hall, 2003.