INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
BAHIA
Campus Salvador

Principais Aspectos de Sistemas Operacionais

Arquitetura de Computadores e Software Básico

Flávia Maristela (flavia@flaviamaristela.com)

O que veremos nesta aula?

- Tudo o que já vimos antes...
  - Visão interna de um sistema operacional
  - Processos e *Threads*
  - Gerência de Memória
  - Arquivos
  - Entrada e Saída

Mas afinal, para que serve um sistema operacional?

Gerenciando o hardware (-- processador --)

- O que faz o processador?
  - Ele é cérebro do computador!
- Ele busca suas instruções para execução na memória
- Instruções podem ser executadas de duas formas:
  - Modo *kernel*
  - Modo usuário

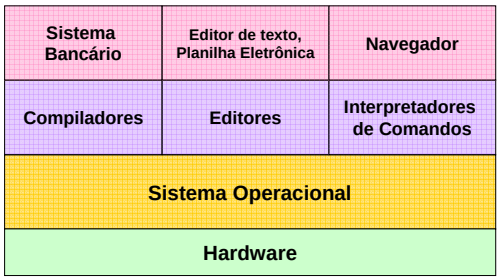
Gerenciando o hardware (-- processador --)

- Modo *kernel*
  - CPU pode executar qualquer instrução
  - CPU tem acesso direto ao hardware
- Modo usuário
  - Permite a execução de apenas um subconjunto de instruções
  - Acesso limitado aos atributos das instruções
  - Funções que envolvem E/S e proteção de memória não são acessíveis

Para pensar um pouco...

- Existe algum programa que precisa executar no modo *kernel*?
- Que programas executam no modo usuário?
- Como um programa do modo usuário pode acessar os dispositivos de E/S?

Como o usuário acessa o hardware?



- Programas de sistema ou utilitários
  - Controle permanece com o usuário
- Chamadas de sistema (*system calls*)
  - Controle é passado para o sistema operacional

Processos

- O que é um processo?
- Programa *vs.* Processos
  - Processo é uma atividade de um programa
- Exemplo: vamos fazer um bolo!
  - Fazer o bolo corresponde a fazer um programa! Então, quais os principais elementos de um programa?

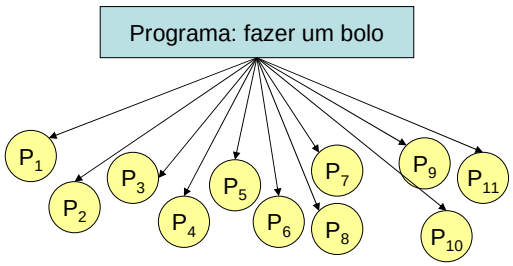
Vamos considerar um exemplo

- Vamos fazer um bolo!

ENTRADA (-- ingredientes --) → PROCESSAMENTO (-- algoritmo --) → SAÍDA (-- bolo --)

1. Bater as claras em neve
2. Misturar açúcar e manteiga
3. Adicionar as gemas
4. Adicionar metade da farinha de trigo
5. Adicionar metade do leite
6. Adicionar a outra metade da farinha de trigo
7. Adicionar a outra metade do leite
8. Ligar o forno a 180°
9. Adicionar as claras
10. Despejar bolo na forma
11. Colocar a forma no forno

Vamos considerar um exemplo

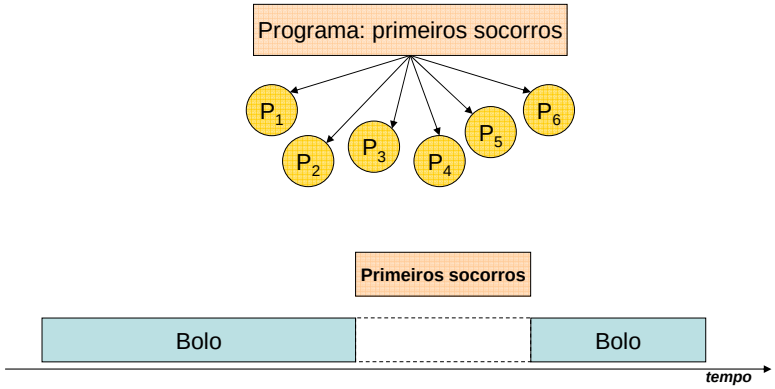


Estados de um processo

- Clássicos
  - Execução
  - Bloqueado
  - Pronto (Apto)
- Outros estados da literatura
  - Iniciado
  - Finalizado (terminado)

Voltando ao nosso exemplo...

Enquanto o bolo está sendo preparado, o filho do cozinheiro cai da bicicleta e se machuca! O que fazer?
O cozinheiro vai **interromper** o bolo para dar os primeiros socorros ao filho!



Voltando ao nosso exemplo...

- Estados dos processos:
  - Em execução:
    - Selecionar ingredientes
    - Misturar açúcar e manteiga
    - Fazer curativo
  - Bloqueado
    - Acabou o leite
    - Faltou algodão
  - Apto
    - O bolo pode ser finalizado enquanto ele atende o filho

- Mas afinal, quem é que coloca ordem na casa?
  - ESCALONADOR: implementação de baixo nível do sistema operacional que controla o acesso a CPU
  - Em nosso exemplo:
    - ESCALONADOR: a consciência do cozinheiro
    - PROCESSADOR: cozinheiro
    - PROCESSOS: atividades que ele precisava realizar

Será que pode piorar?

Imagine agora uma padaria, onde além do bolo é necessário fazer pães, salgados, doces...

compartilhando os mesmos ingredientes

Isso se chama PROGRAMAÇÃO CONCORRENTE

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
BAHIA
Campus Salvador

Processos

Porque eu preciso entender este assunto?

- Para entender como um computador consegue executar várias tarefas simultaneamente e **qual o impacto que isso pode ter em meus programas**!

Os programas de ontem...

- Antigamente, os computadores eram máquinas dedicadas:
  - Possuíam apenas um usuário
  - Executavam apenas um programa por vez
    - Programas em execução tinham total controle dos recursos do computador

E os programas de hoje!

- Hoje os computadores:
  - Executam vários programas simultaneamente
  - Podem ser usados por vários usuários
- Isso gerou a necessidade de compartilhar recursos...
- ... e por isso os programas foram divididos em unidades menores.

Sobre os programas...

- Quando ligamos o computador, vários programas começam a ser executados.
  - Programas ativados pelo Sistema Operacional
  - Programas ativados pelo usuário
- Cada um destes programas possui vários ***processos***.

Processos (-- definição --)

- Definição:
  - Programa em execução

Silberschatz, Tanenbaum

- Processos são entidades independentes entre si, mas *concorrem* aos mesmos recursos do computador.

Processos (-- estados --)

- Novo
  - O processo está sendo criado, ou seja, seu código está sendo carregado em memória, junto com suas bibliotecas;
  - As estruturas de dados do *kernel* estão sendo atualizadas para permitir sua execução.
- Pronto
  - Processo está em memória, pronto para ser executado, aguardando a disponibilidade do processador;
  - IMPORTANTE: Os processos “prontos” são organizados em uma fila cuja ordem é determinada por algoritmos de escalonamento.

Processos
(-- estados --)

- Executando:
  - Processo está executando suas instruções.
- Bloqueado
  - Processo não pode executar porque depende de recursos ainda não disponíveis (dados, algum tipo de sincronização, a liberação de algum recurso compartilhado);
  - Processo simplesmente espera o tempo passar (em estado de "*sleeping*").
- Terminado
  - A execução do processo foi encerrada e ele pode ser removido da memória do sistema.

Processos
(-- transições --)

- ... → **Novo**
  - um novo processo é criado e começa a ser preparado para executar.
- **Novo → Pronto**
  - o novo processo termina de ser carregado em memória, estando pronto para executar.
- **Pronto → Executando**
  - o processo é escolhido pelo escalonador para ser executado, entre os demais processos prontos.

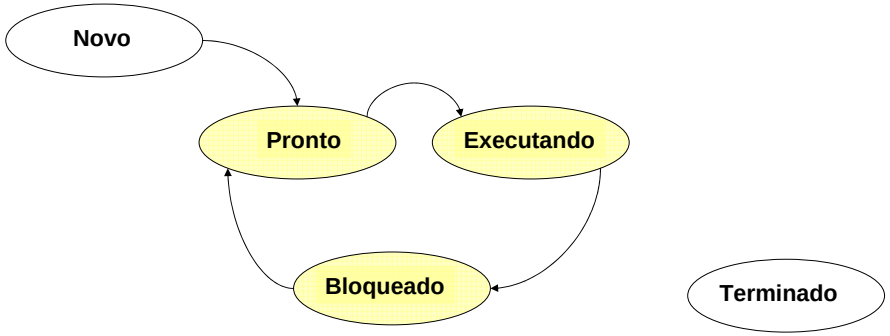
Processos
(-- transições --)

- **Executando → Pronto**
  - esta transição ocorre quando se esgota a fatia de tempo destinada ao processo (***quantum***);
  - Nesse momento o processo não precisa de outros recursos além do processador e por isso volta à fila de "pronto" para esperar novamente a disponibilidade do processador.
- **Executando → Terminada**
  - O processo encerra sua execução ou é abortado em consequência de algum erro (acesso inválido à memória, instrução ilegal, divisão por zero).
  - Em geral, o processo que deseja terminar avisa ao sistema operacional através de uma chamada de sistema.

Processos
(-- transições --)

- **Terminado → ...**
  - Quando terminado, um processo é removido da memória e seus registros e estruturas de controle no *kernel* são apagados.
- **Executando → Bloqueado**
  - caso o processo em execução solicite acesso a um recurso não disponível, ele abandona o processador e fica bloqueado até o recurso ficar disponível.
- **Bloqueado → Pronto**
  - quando o recurso solicitado pelo processo se torna disponível, ele pode então voltar ao estado de "pronto".

Processos
(-- transições --)



Processos
(-- execução --)

- Os processos podem executar de duas formas:
  - Em *FOREGROUND*
    - Processos que interagem com os usuários
  - EM *BACKGROUND*
    - Não associados a usuários
    - Possuem funções específicas

Processos
(-- finalização --)

- O que motiva a finalização de um processo?
  - Saída normal;
  - Saída com erro;
  - *Fatal Error* (involuntário);
  - Outro processo (involuntário)

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
BAHIA
Campus Salvador

Threads

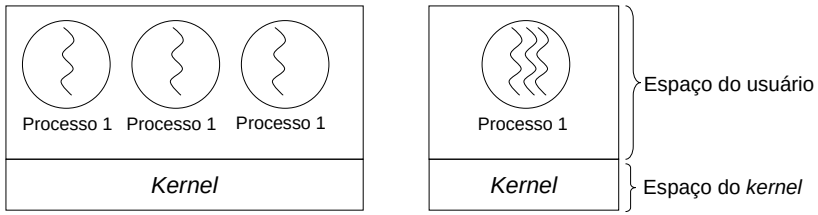
THREADS

- Motivação:
  - A necessidade de compartilhar diferentes recursos do computador deu origem a *PROGRAMAÇÃO CONCORRENTE*.
  - Neste cenário, um programa que tinha vários processos com um único fluxo de execução passou a ter vários processos.
  - Cada processo possuía um ou mais fluxos de execução.

Threads

- Definição:
  - "Entidades escalonadas para execução"
    Tanenbaum
  - "Fluxo de execução dentro de um processo"
    Rômulo Oliveira
  - "Unidade básica de utilização da CPU"
    Silbershatz
- *Threads* compartilham os mesmos recursos de um processo

Visão clássica *vs.* Visão contemporânea



- Quando usamos as *threads*?
  - Quando precisamos de *Programação concorrente*!

THREADS

- *Multithreading*
  - Termo usado para caracterizar um processo com várias *threads.*
  - Sistema **multithread** executa as *threads* tão rapidamente, que passa ao usuário a impressão de que as mesmas estão sendo executadas em paralelo.
  - O termo também está ligado a dispositivos de hardware que permitem a execução de várias *threads*.

PARA PENSAR! Qual é a diferença entre os seguintes termos:
-Paralelismo
-Pseudo-paralelismo
-Multiprogramação
-Multithreading
-Time sharing

THREADS

- *Threads* compartilham os recursos de um processo;
- *Threads* de um mesmo processo não são independentes entre si
- Em sistemas *multithread*, normalmente cada processo inicia com apenas uma *thread*
  - Esta *thread* tem a capacidade de criar novas *threads*

THREADS

- *Threads* não representam a solução para todos os problemas:
  - Se um processo é duplicado, ele deve manter todas as *threads* do processo pai?
  - Se uma *thread* estava bloqueada no momento da cópia de um processo, a *thread do processo* filho também vai estar?
  - Quando um dado é útil para uma *thread*, quem vai receber uma cópia, apenas o processo pai? O processo filho também deve receber?

THREADS

- *Threads* podem ser gerenciadas em dois níveis:
- Nível do usuário
- Nível do *kernel*

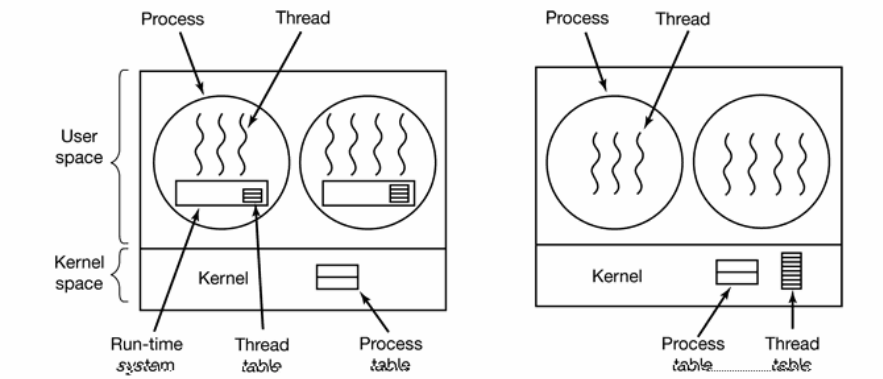
THREADS
(-- nível do usuário --)

- *Kernel* do sistema operacional não tem conhecimento sobre tais *threads*.
- Sistema operacional enxerga apenas um único processo com uma única *thread.*

THREADS
(-- nível do kernel --)

- *Kernel* do sistema operacional controla todas as operações entre *threads:*
  - *Create*
  - *Terminate;*
  - *Join*
  - *Yield*
  - *Resource sharing* (compartilhamento de recursos)

THREADS
(-- nível usuário *vs.* nível kernel --)



Dúvidas?