Laboratório de Pesquisa em Redes e Multimídia

Sistemas Operacionais

UNIX: processos e o kernel mode

Universidade Federal do Espírito Santo
Departamento de Informática

O Kernel

- É um programa especial, uma parte privilegiada do sistema operacional, responsável por funções críticas do S.O., como escalonamento da UCP e tratamento de interrupções.
  - Ele implementa o modelo de processos do sistema
  - O kernel não é um processo!
- O kernel oferece uma série de serviços
- No Unix, o código do kernel reside em um arquivo em disco
  - *no Ubuntu, /boot/vmlinuz...*
- Programa *bootstrapping* carrega o kernel para a memória na inicialização do sistema.

A Abstração "Processo"

- Processo é uma entidade que **executa** um programa e que provê um **ambiente de execução** para ele
- Processos têm um ciclo de vida:
  - São criados pelas SVCs ***fork*** ou ***vfork*** e rodam até que sejam terminados (por meio da primitiva *exit*).
- Durante a sua execução um mesmo processo pode rodar um ou mais programas
  - A primitiva ***exec*** é invocada para rodar um novo programa.
- Processos no Unix possuem uma **hierarquia**
  - Cada processo tem um processo pai (*parent process*) e pode ter um ou mais processos filhos (*child processes*).

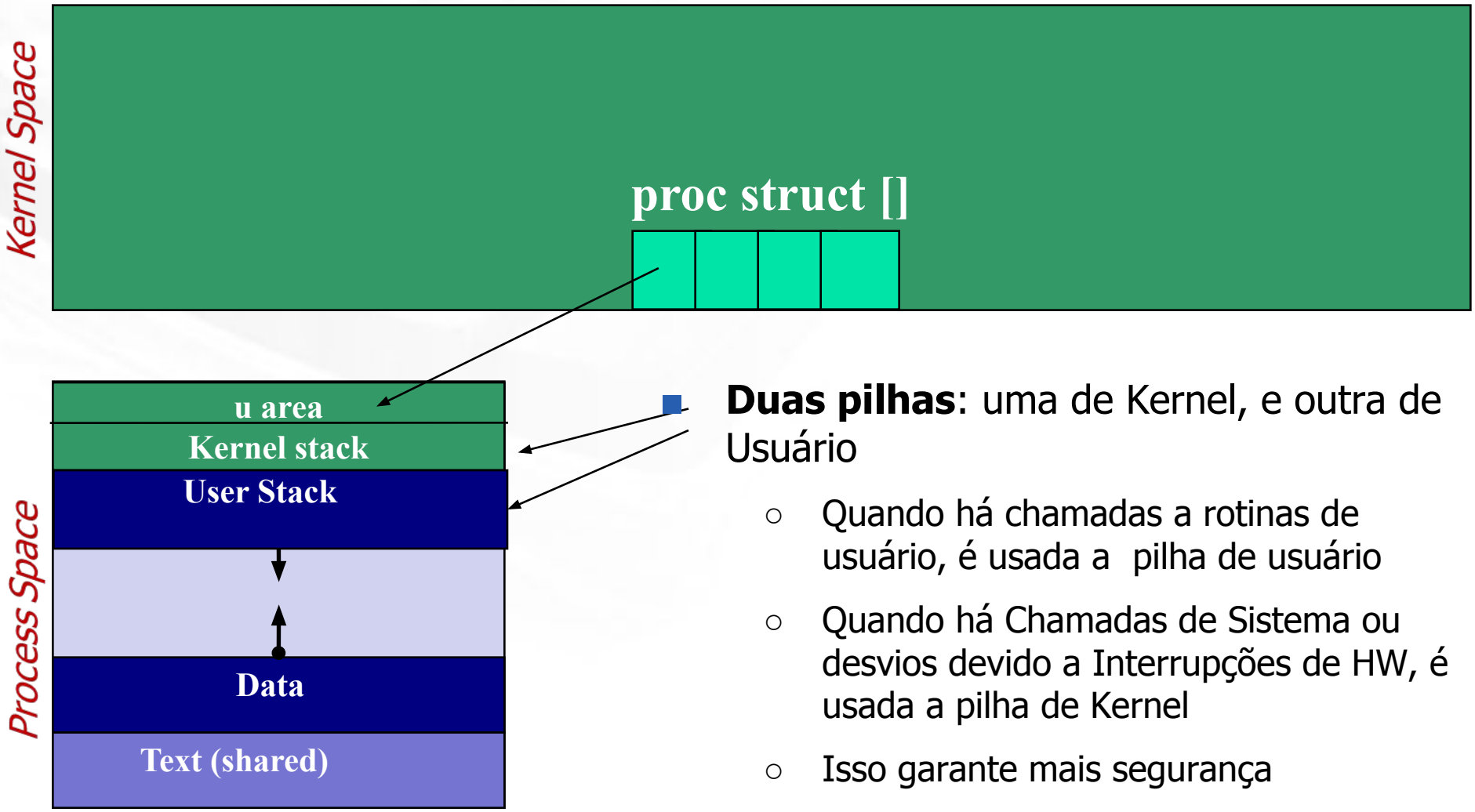
A Abstração "Processo" (cont.)

- O processo ***init*** **(PID 1)** localiza-se no topo da hierarquia de processos de usuário do Unix.
  - Primeiro processo de usuário a ser criado, quando o sistema é iniciado.
  - Executa o programa */etc/init* ou */sbin/init*
- Todos os processos descendem do processo *init*, à exceção de alguns poucos, como o *swapper* ou o ***kthreadd*** *(Linux)*
  - Esses também são criados na inicialização do sistema e ajudam o kernel na execução das suas tarefas
  - Diferentemente do *init*, esses processos são implementados dentro do próprio kernel, ou seja, não há um programa (arquivo) binário regular para eles
- **Se, ao terminar, um processo tiver processos filhos ativos, estes tornam-se órfãos e são herdados pelo processo *init*.**

Processos e Memória Virtual

- A maioria das implementações do UNIX usa memória virtual
  - Endereços utilizados no programa não referenciam diretamente memória física
  - Cada processo possui seu **"espaço de endereçamento virtual"**

System(Kernel) Space x Process Space



- **Duas pilhas**: uma de Kernel, e outra de Usuário
  - Quando há chamadas a rotinas de usuário, é usada a pilha de usuário
  - Quando há Chamadas de Sistema ou desvios devido a Interrupções de HW, é usada a pilha de Kernel
  - Isso garante mais segurança

Estruturas de Controle do Unix

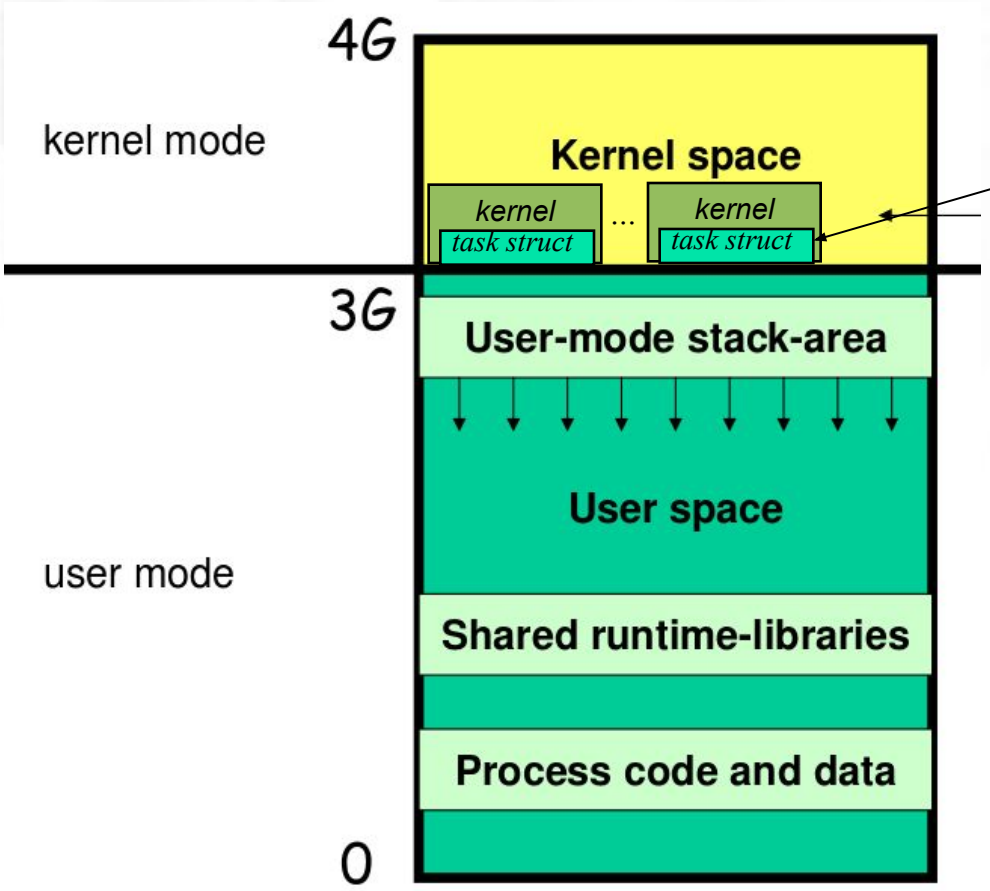
O "Bloco de Controle" no Unix é dividido em 2 partes:

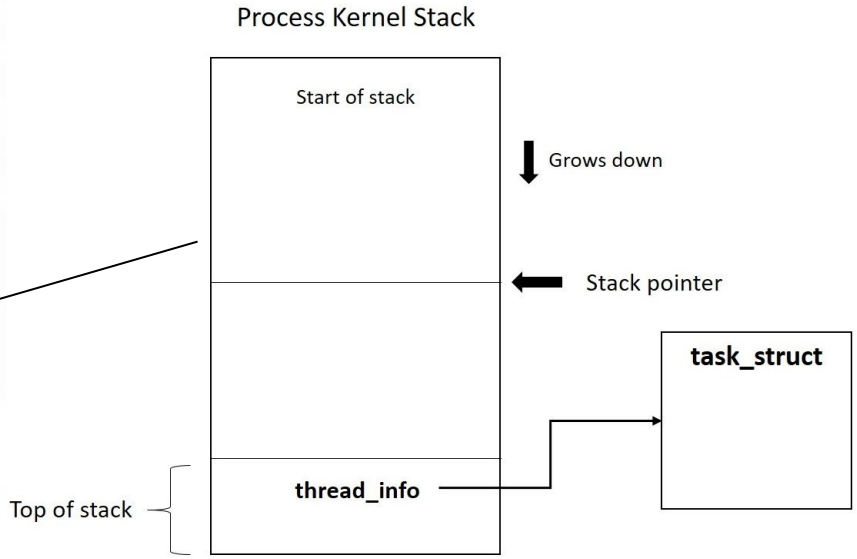
- **Proc Structure**
  - Contém todas as informações que o kernel posssa precisar quando um processo **NÃO está *running***
  - Mesmo que o processo vá para disco (susppended), sua Proc Struct permanece em memória!
  - Encontra-se no *system (kernel) space*
- **u Area**
  - Contém informações necessárias apenas quando o processo está *running*, ou vai entrar nesse estado
  - Encontra-se no *process space*

No Linux é diferente...

A partir da versão 2.6, mudança na Kernel Stack

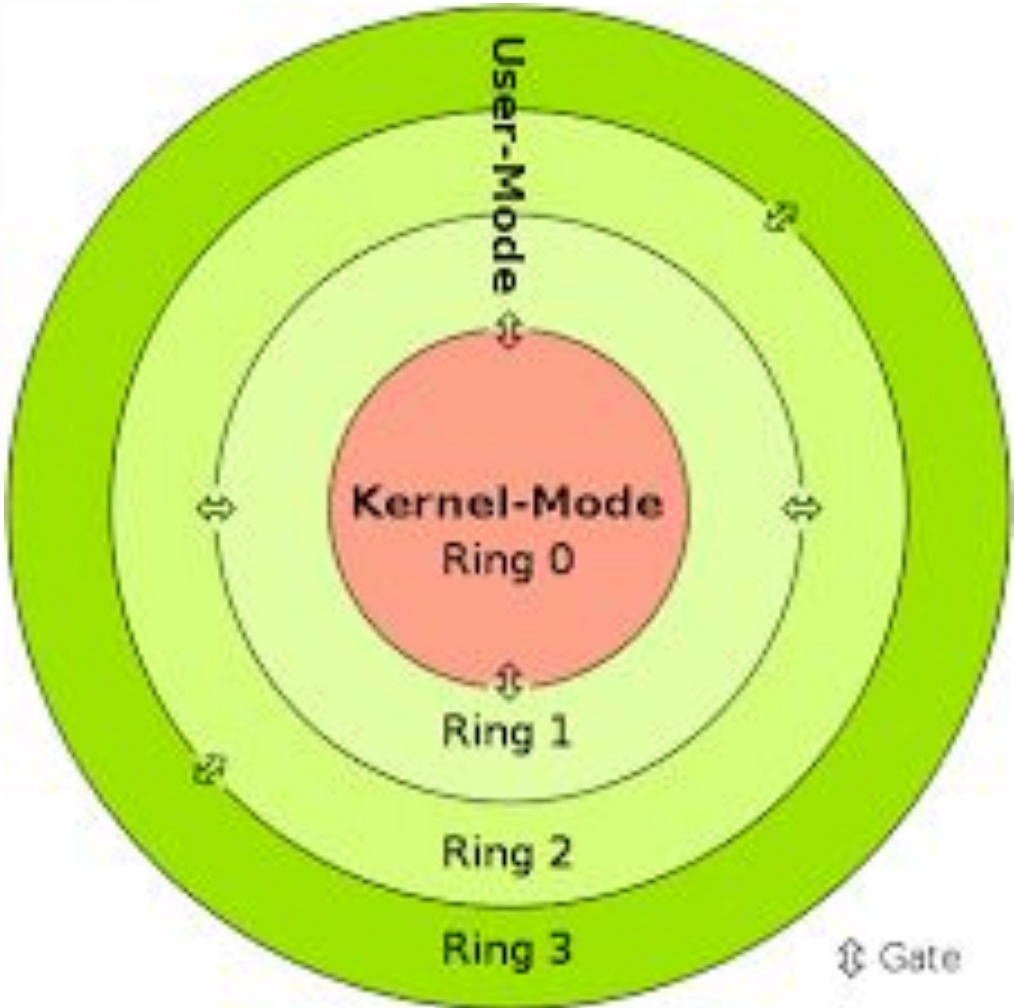
Antes da versão 2.6





Modos de Operação da CPU

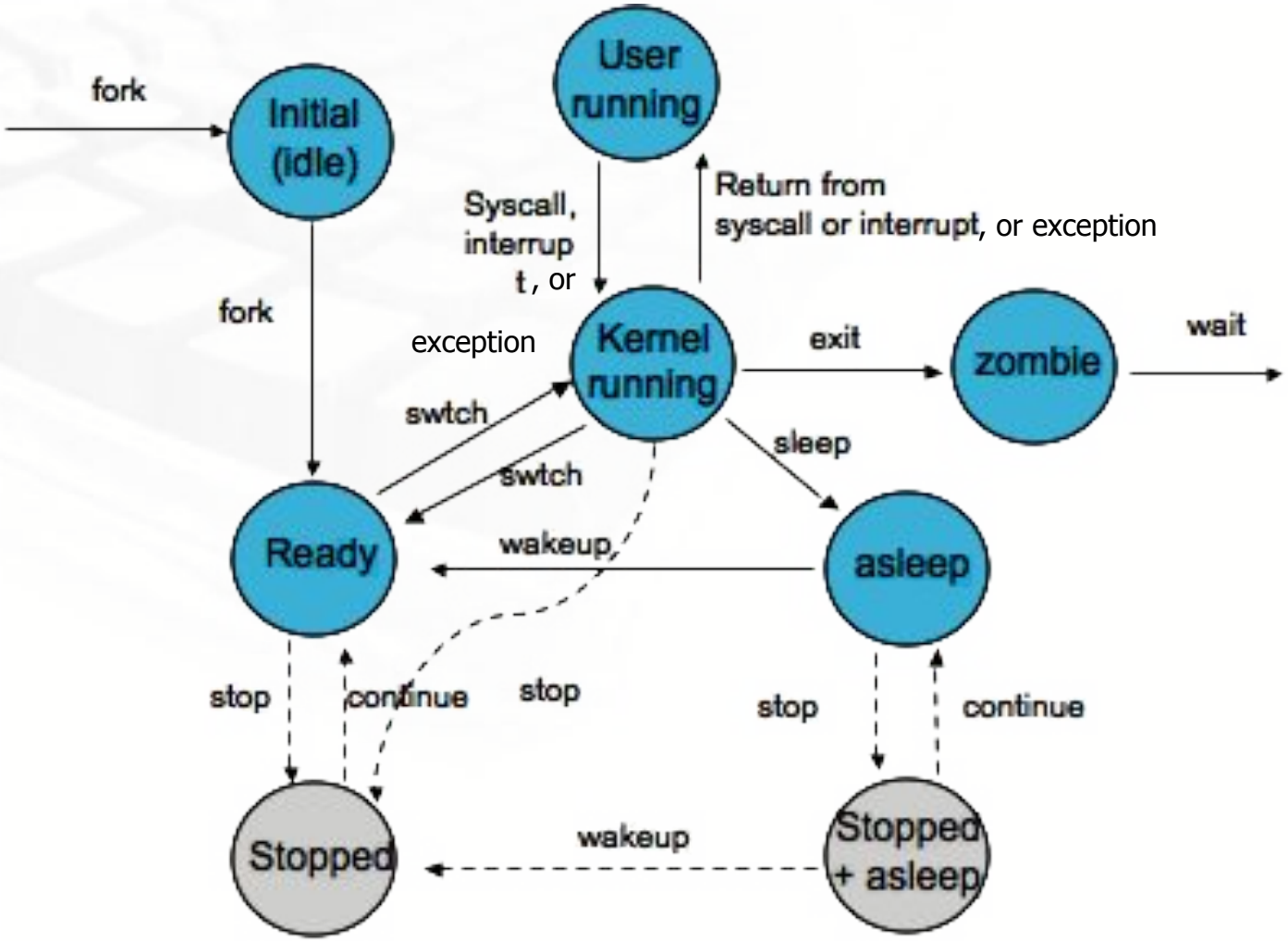
- Em geral, CPUs oferecem 4 modos de operação:



User Mode x Kernel Mode

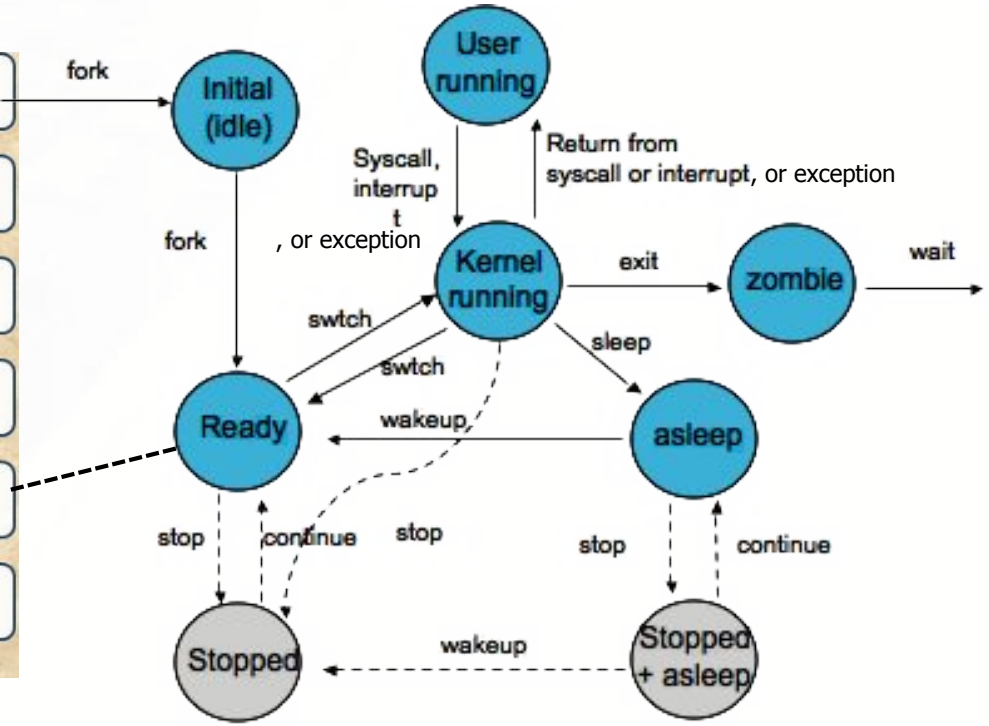
- Com finalidade principal de **proteção**, a maioria das CPUs atuais fornece pelo menos dois modos de operação.
- O Unix requer do hardware a implementação de apenas 2 modos de operação:
  - *user mode* (menos privilegiado)
  - *kernel mode* (com mais privilégios).
- Processos de usuário rodam em *user mode*
  - Processos não podem – acidental ou maliciosamente – corromper outro processo ou mesmo o kernel
- Acesso ao *kernel space* só pode ser feito em *kernel mode*
  - O processo só consegue acessar o *kernel space* via SVC (portanto, acesso controlado).

User Mode/Kernel Mode x Máquina de Estados



... Quando um processo é criado (**fork()**)....

1. Allocate a slot in the process table for the new process
2. Assign a unique process ID to the child process
3. Make a copy of the process image of the parent, with the exception of any shared memory
4. Increments counters for any files owned by the parent, to reflect that an additional process now also owns those files
5. Assigns the child process to the Ready to Run state
6. Returns the ID number of the child to the parent process, and a 0 value to the child process

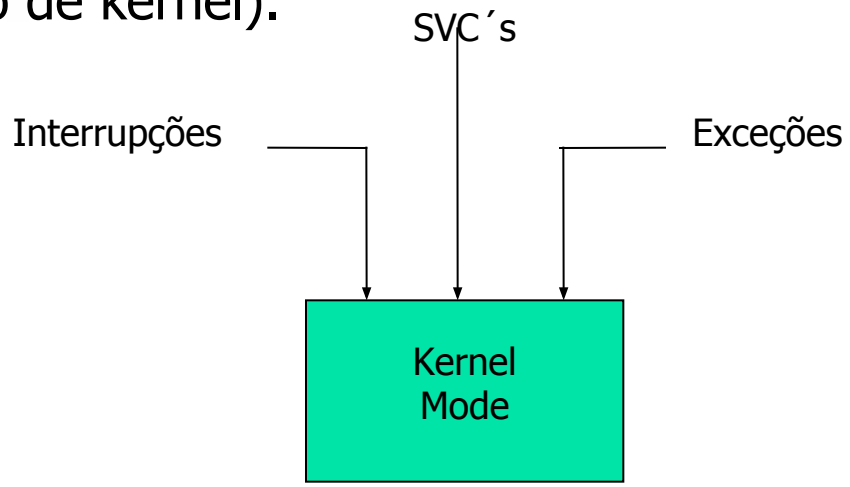


Executando em Kernel Mode

- Existem **3 tipos de eventos** que fazem o sistema (a CPU) entrar em *kernel mode*:
  - As chamadas ao sistema (SVCs/traps/interrupções de SW)
  - As interrupções de HW (provenientes dos dispositivos periféricos)
  - As exceções
- Na ocorrência de uma delas, uma sequência especial de instruções é executada para colocar a CPU em *kernel mode* e passar o controle para o kernel (desvio para o código de kernel).

HW alterna para kernel mode utilizando a kernel stack do processo:

- o HW salva PC e Status (e possivelmente outros "estados" na kernel stack)
- o SW (rotina assembly) salva outras informações necessárias para seguir o tratamento do evento



Executando em Kernel Mode (cont.)

- **Interrupções**
  - Eventos assíncronos causados por dispositivos periféricos (disco, relógio, interface de rede, etc).
  - Quando ocorre uma interrupção, há desvio para o kernel
  - Aqui temos código de kernel sendo executado para realizar gerência do sistema
    - O processo que está rodando, passa de "user running" para "kernel running", mas o código de kernel que passa a ser executado não tem nenhuma relação com o processo em si

Executando em Kernel Mode (cont.)

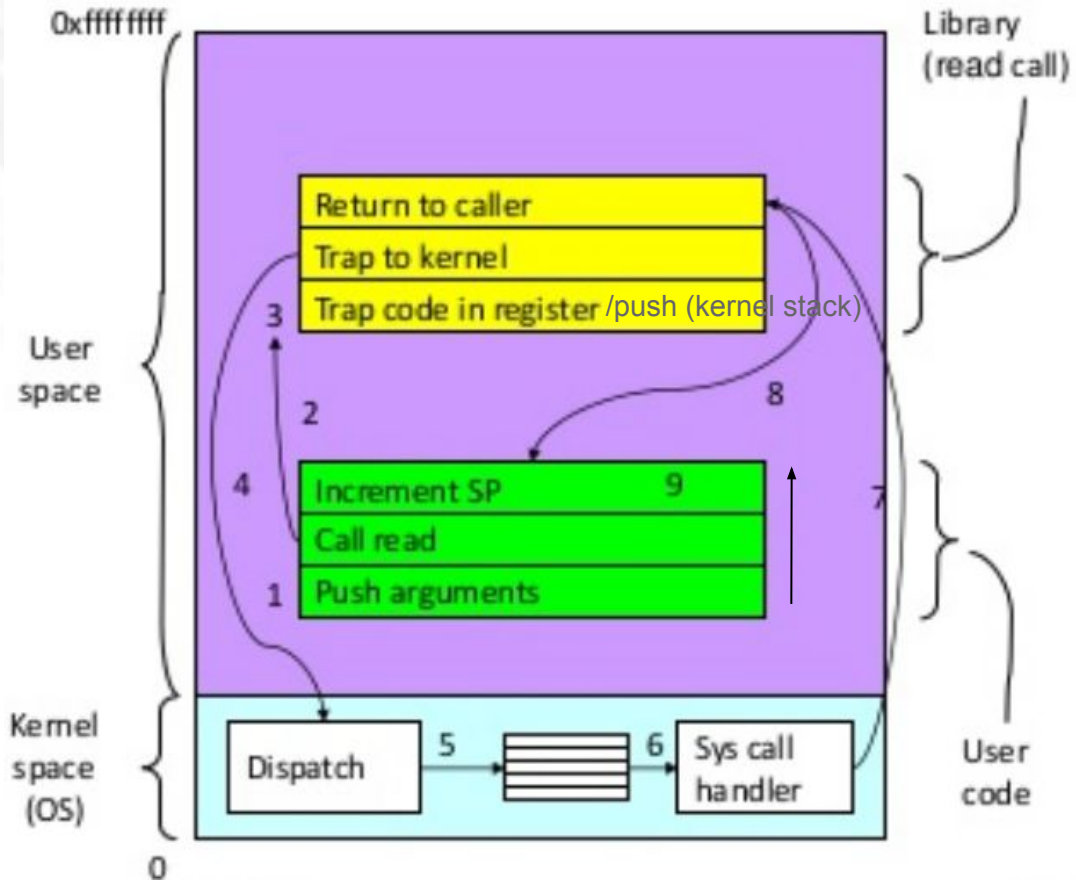
- **Exceções**
  - Eventos síncronos ao processo em execução, causados por eventos relacionados ao próprio processo
    - Divisão por zero, acesso a endereço ilegal, *overflow*, etc.)

Obs: são ditos "síncronos" pois , se executarmos um mesmo código com as mesmas condições e dados de entradas, as mesmas exceções serão causadas

Executando em Kernel Mode (cont.)

- **Supervisor Calls (SVCs)**
  - Ocorrem quando o processo executa uma instrução especial do processador (*trap*, *syscall*)
  - São eventos **síncronos** ao processo em execução
  - Assim como as exceções, a SVC roda em ***process context***:
    - Pode acessar o espaço de endereçamento do processo (incluindo *u area*), além de poder bloquear o processo, se necessário.

Execução de uma SVC no UNIX

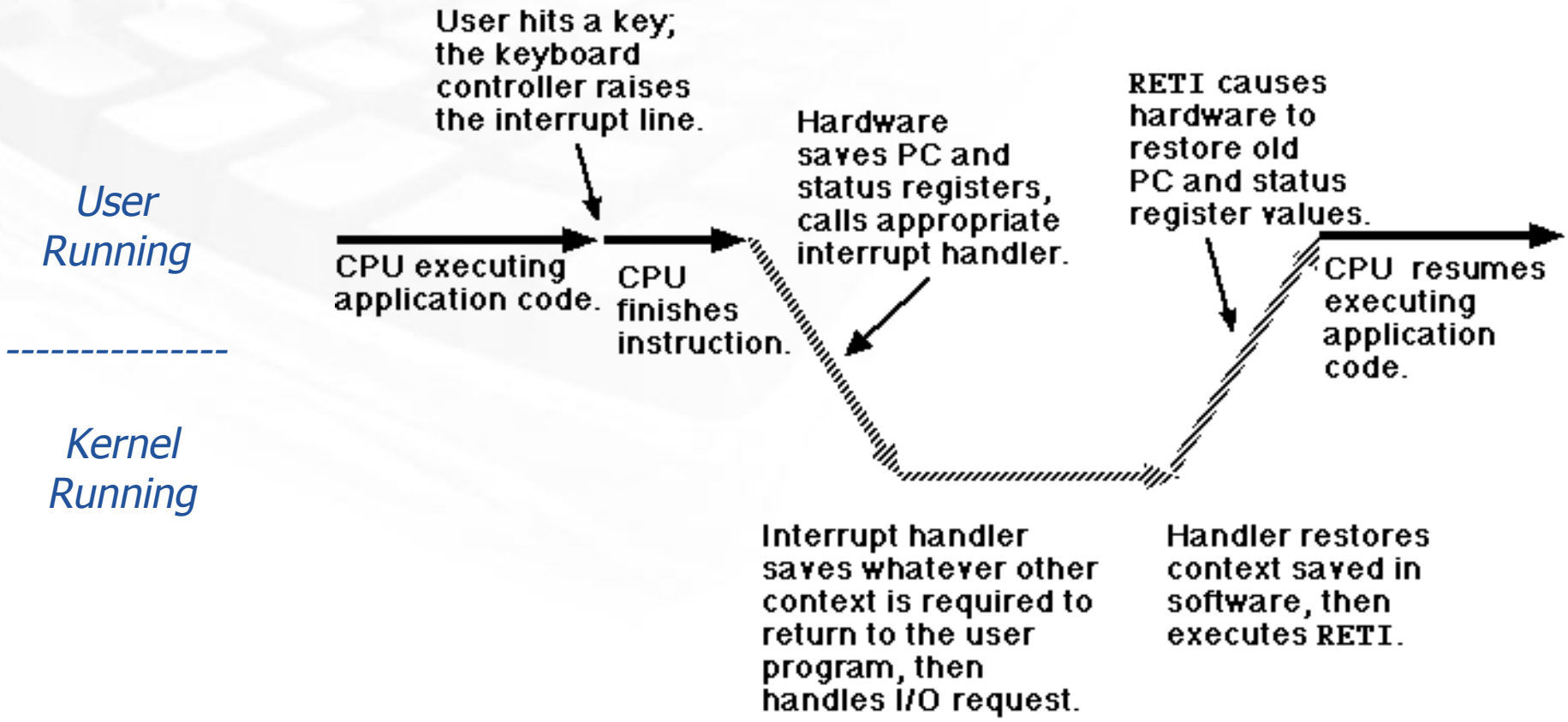


- System call: read(fd,buffer,length)
- Program pushes arguments, calls library
- Library sets up trap, calls OS
- OS handles system call
- Control returns to library
- Library returns to user program

Manipulação de Interrupções

- Interrupções são eventos gerados assincronamente à atividade regular do sistema.
  - O sistema não sabe em que ponto no fluxo de instruções a interrupção ocorrerá.
- *Interrupt Handler* é o nome dado à rotina de tratamento da interrupção.
- O Interrupt Handler roda em *kernel mode*
  - Ali são executadas tarefas do kernel (de gerência do sistema) que não têm relação com o processo que se encontra "running" naquele instante ...
  - ... apesar disso... o tempo de servir a interrupção é descontado do quantum do processo em execução (*time-slice*).

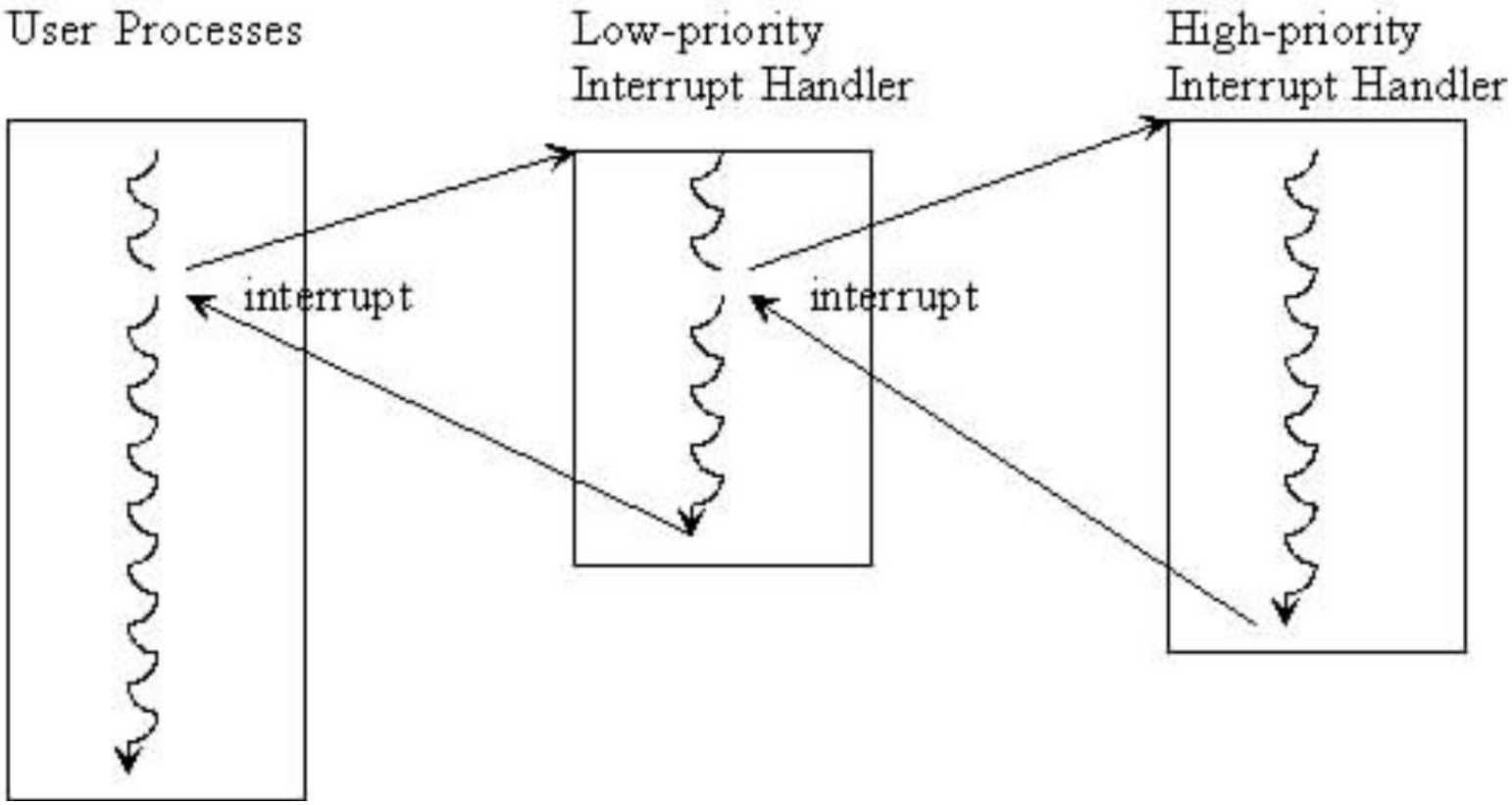
Manipulação de Interrupções (cont.)



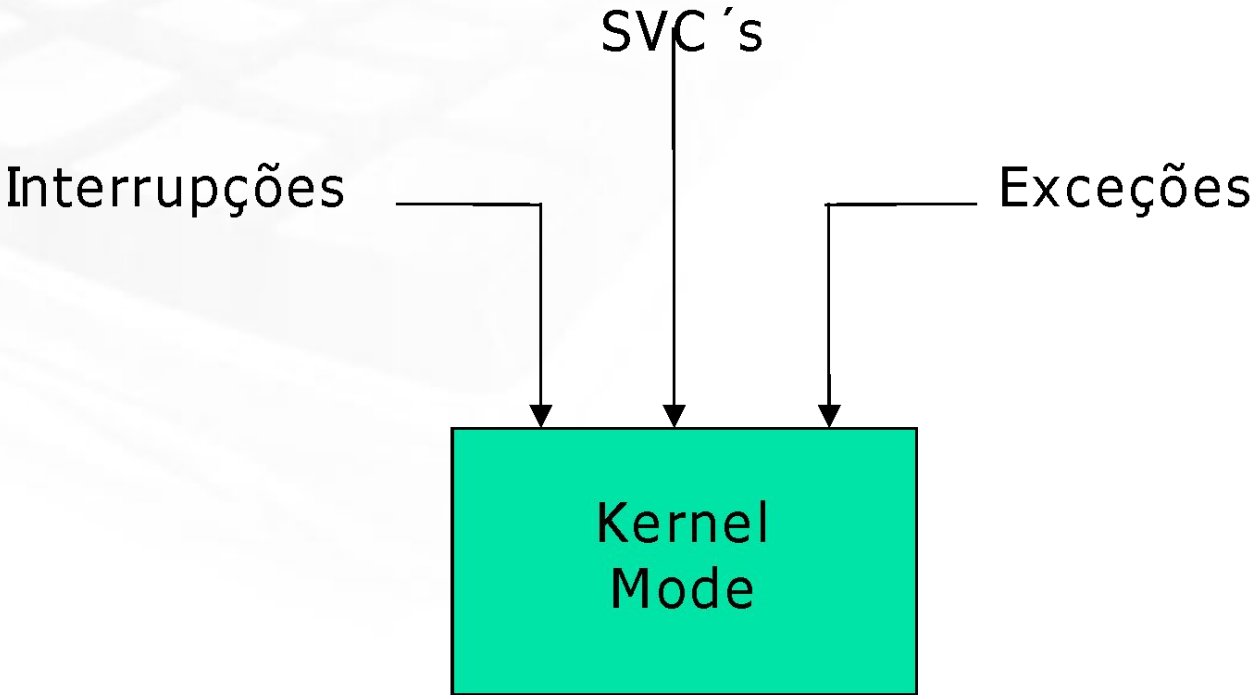
Manipulação de Interrupções - IPL

- Interrupções podem ocorrer enquanto uma outra está sendo atendida
  - Necessidade de se **priorizar as interrupções**
  - Ex: interrupção de relógio é mais prioritária do que a interrupção de interface de rede (cujo processamento dura vários *ticks* do relógio)
- A cada tipo de interrupção está associado a um ***Interrupt Priority Level - IPL***
  - O número de níveis de interrupção varia de acordo com o padrão Unix e com as arquiteturas de hardware.
  - Ex: Unix BSD – IPL´s variam de 0 a 31.

Manipulação de Interrupções - IPL



Kernel Mode.... Recapitulando!!!



Referências

- VAHALIA, U. Unix Internals: the new frontiers. Prentice-Hall, 1996
  Capítulo 2 (até seção 2.4)