Sistemas Operacionais I

Profa. Kalinka Regina Lucas Jaquie Castelo Branco
kalinka@icmc.usp.br

Universidade de São Paulo

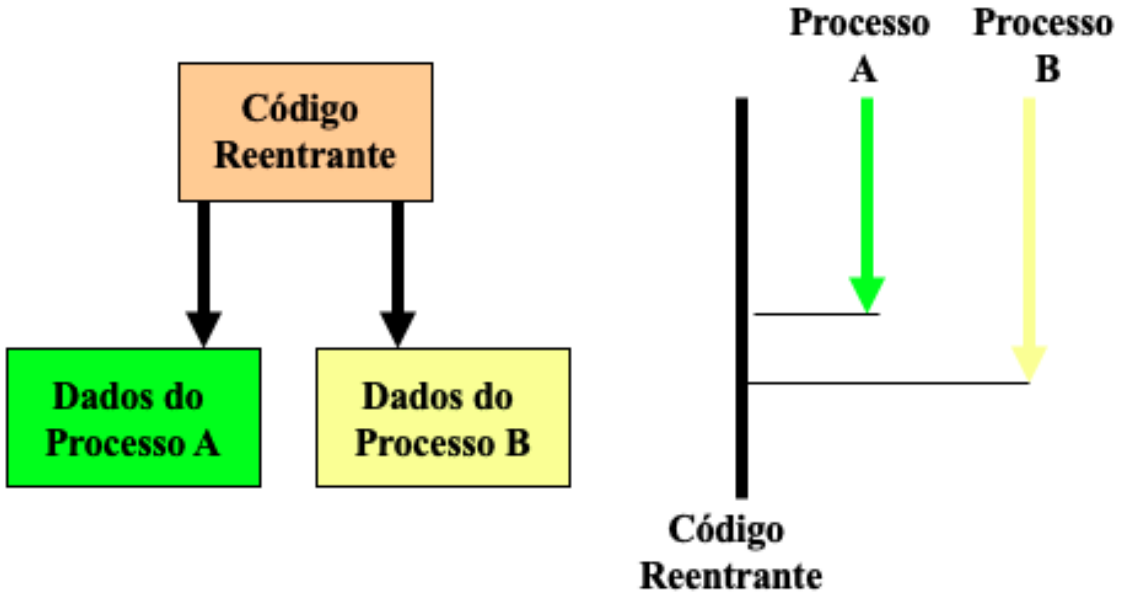
Setembro de 2020

Processos

- Definição de Processo
  - É um programa em execução, incluindo os valores correntes do controlador de programa, registradores e variáveis.
  - O conceito de processo é dinâmico, em contraposição ao conceito de programa, que é estático.
  - Nem sempre um programa equivale a apenas um processo.
  - Em sistemas que permitem reentrância, o código de um programa pode gerar vários processos.

Reentrância

- Capacidade de um código executável (reentrante) ser compartilhado por diversos usuários, exigindo que apenas uma cópia do programa esteja na memória.
- Permite que cada usuário possa estar em um ponto diferente do código reentrante, manipulando dados próprios, exclusivos de cada usuário.



Processo

- Caracterizado por programas em execução.
- Cada processo possui:
  - Programa (instruções que serão executadas);
  - Espaço de endereço de memória (mínimo e máximo);
  - Contextos de software: atributos.

O Sistema Operacional gerencia todos os processos fazendo uso do Bloco de Controle de Processos - BCP

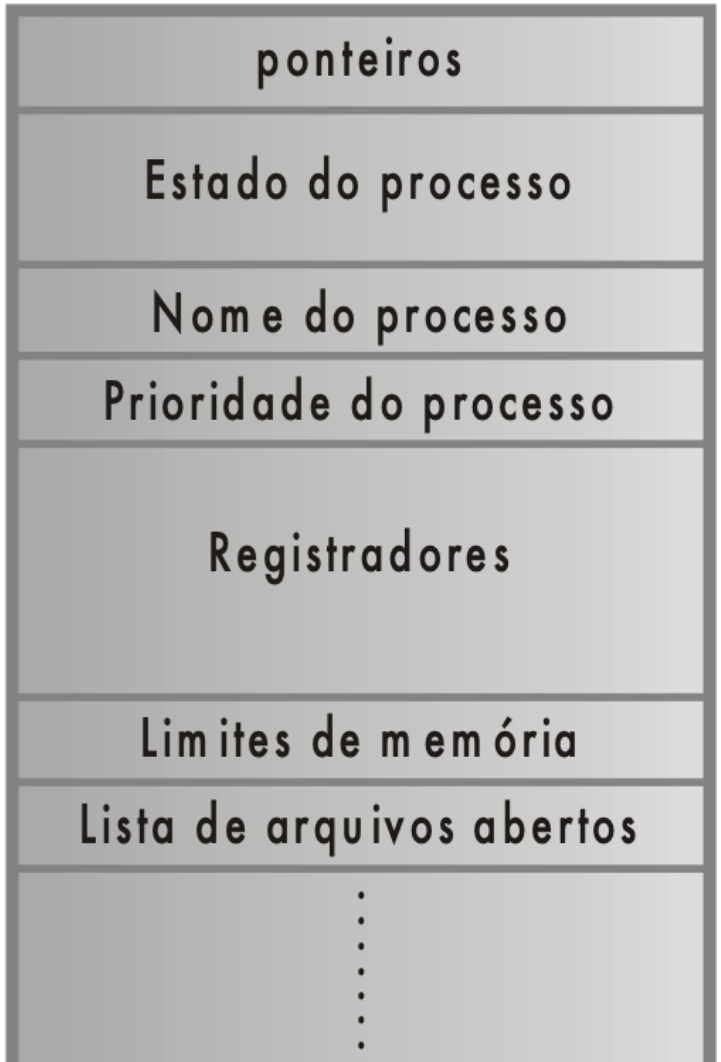
Processo

- Basicamente, um processo possui três segmentos:
  - **Texto**: código do(s) programa(s);
  - **Dados**: as variáveis;
  - **Pilha de execução**: controla a execução do processo.

Processo - Contextos

Processo - Bloco de Controle de Processos

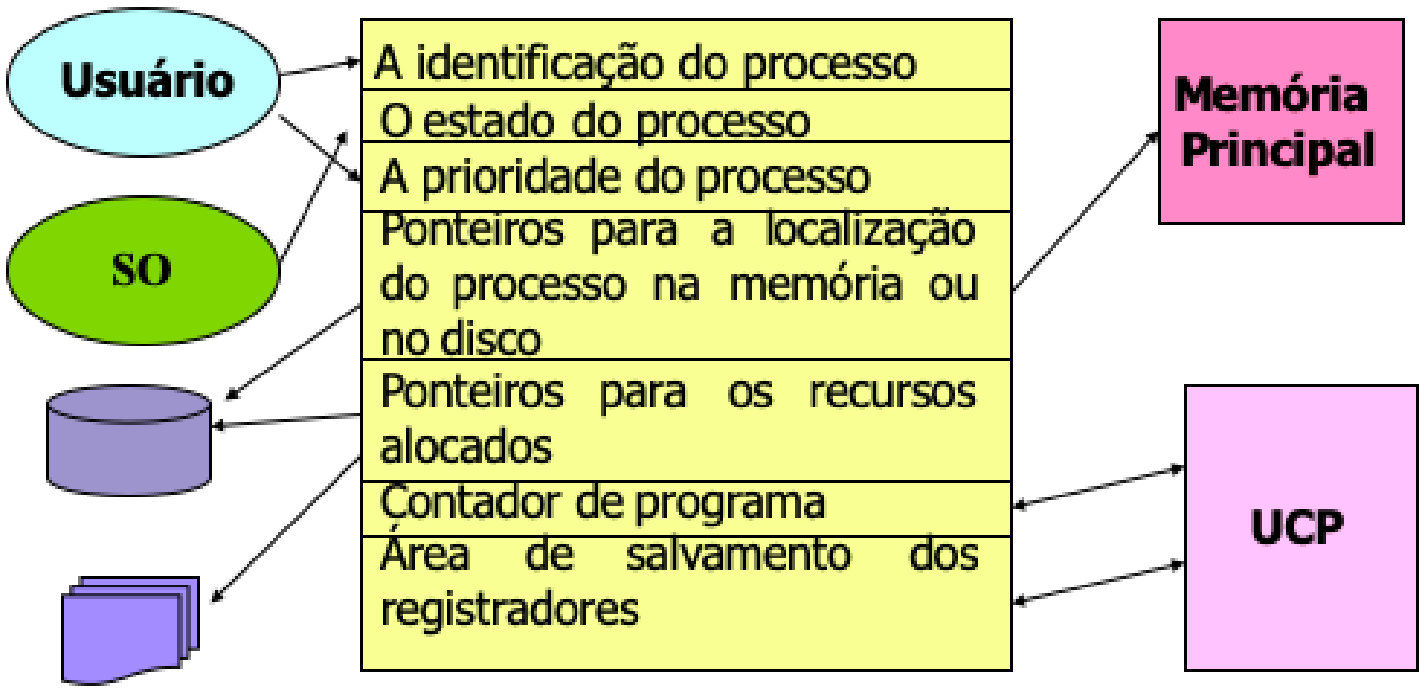
- Bloco de Controle de Processo - Contém informações sobre o processo.



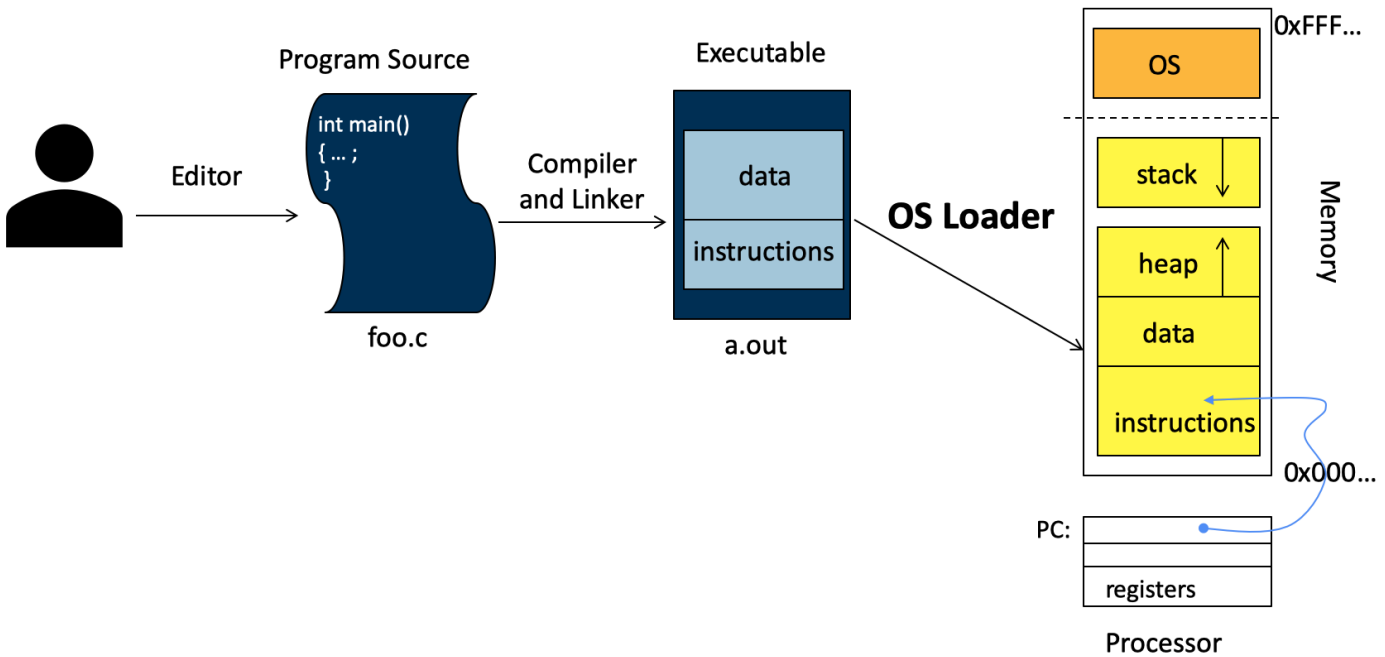
Processo - Bloco de Controle de Processos

- No sistema, cada processo será representado por seu resumo, que consiste no BCP, também conhecido por Bloco de Controle de Programa ou Descritor de Processo. O BCP consiste de uma estrutura de dados contendo informações importantes sobre o processo, incluindo:
  - A identificação do processo;
  - O estado do processo;
  - A prioridade do processo;
  - Ponteiros para a localização do processo na memória ou no disco;
  - Contador de programa;
  - Área de salvamento dos registradores, etc.

Processo - Bloco de Controle de Processos



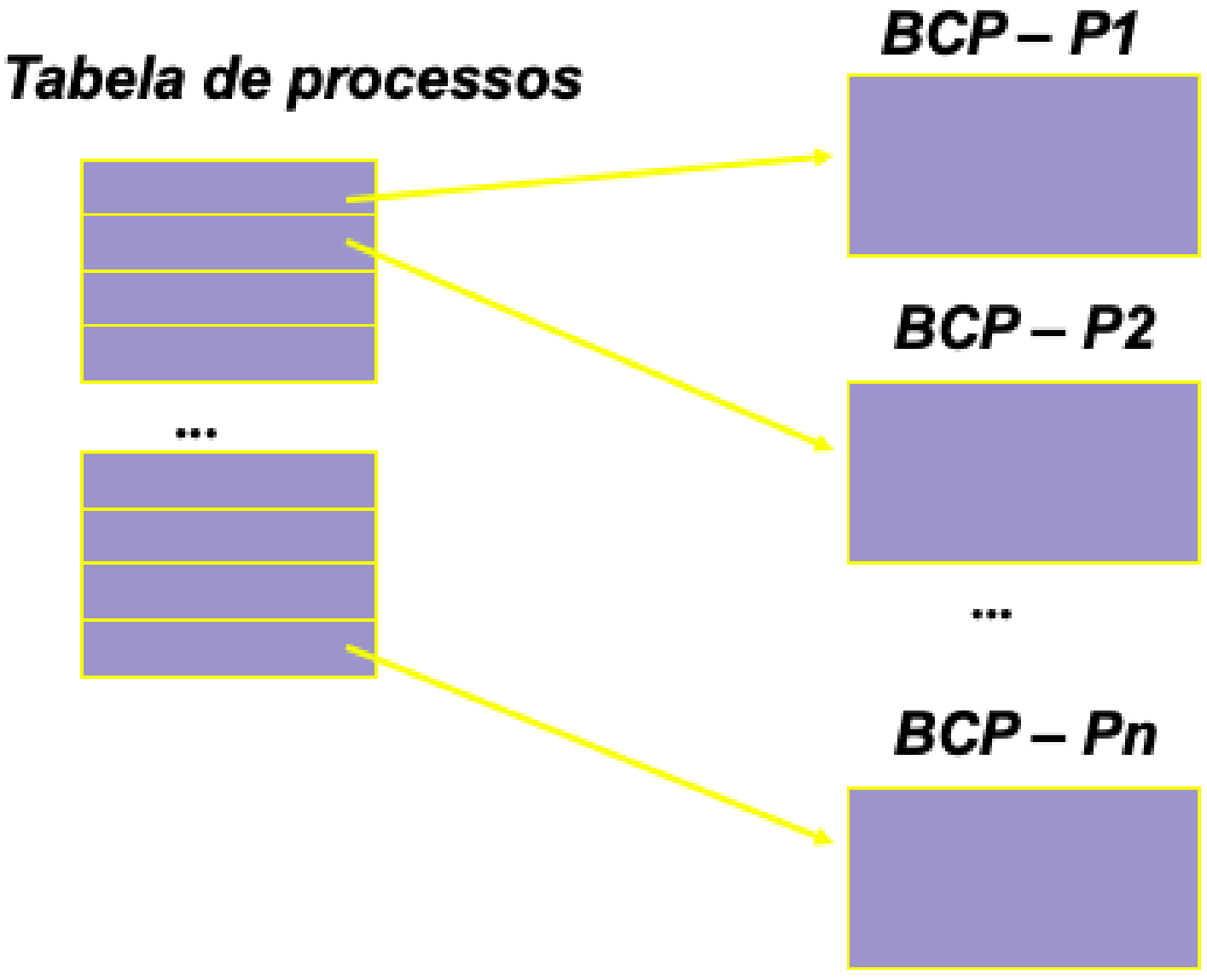
Processo - Resultado do Sistema Operacional: Executar Programas



Processo - Implementação

- Tabela de Processos:
  - Cada processo possui uma entrada;
  - Cada entrada possui um ponteiro para o BCP ou descritor do processo;
  - BCP possui todas as informações do processo - Contextos de hardware, software, endereço de memória.

Processo - Implementação



Processo - Implementação

Algumas informações do BCP

Process management	**Memory management**	**File management**
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Processo - Criação

- Principais eventos que causam a criação de um processo:
  - Inicialização do Sistema;
  - Execução de uma chamada ao sistema de criação de processo por um processo em execução;
  - Uma requisição de Usuário para a criação de um novo processo;
  - Início de um *Job* em lote.

Processo - Criação

- Unix:
  - $FORK$;
    - Cria processo Pai e processo filho com mesmo endereçamento;
    - Depois o processo filho tem endereçamento separado.
- Windows
  - $CreateProcess$
    - Cria processo Pai e processo Filho com mesmo endereçamento sempre.

Processo

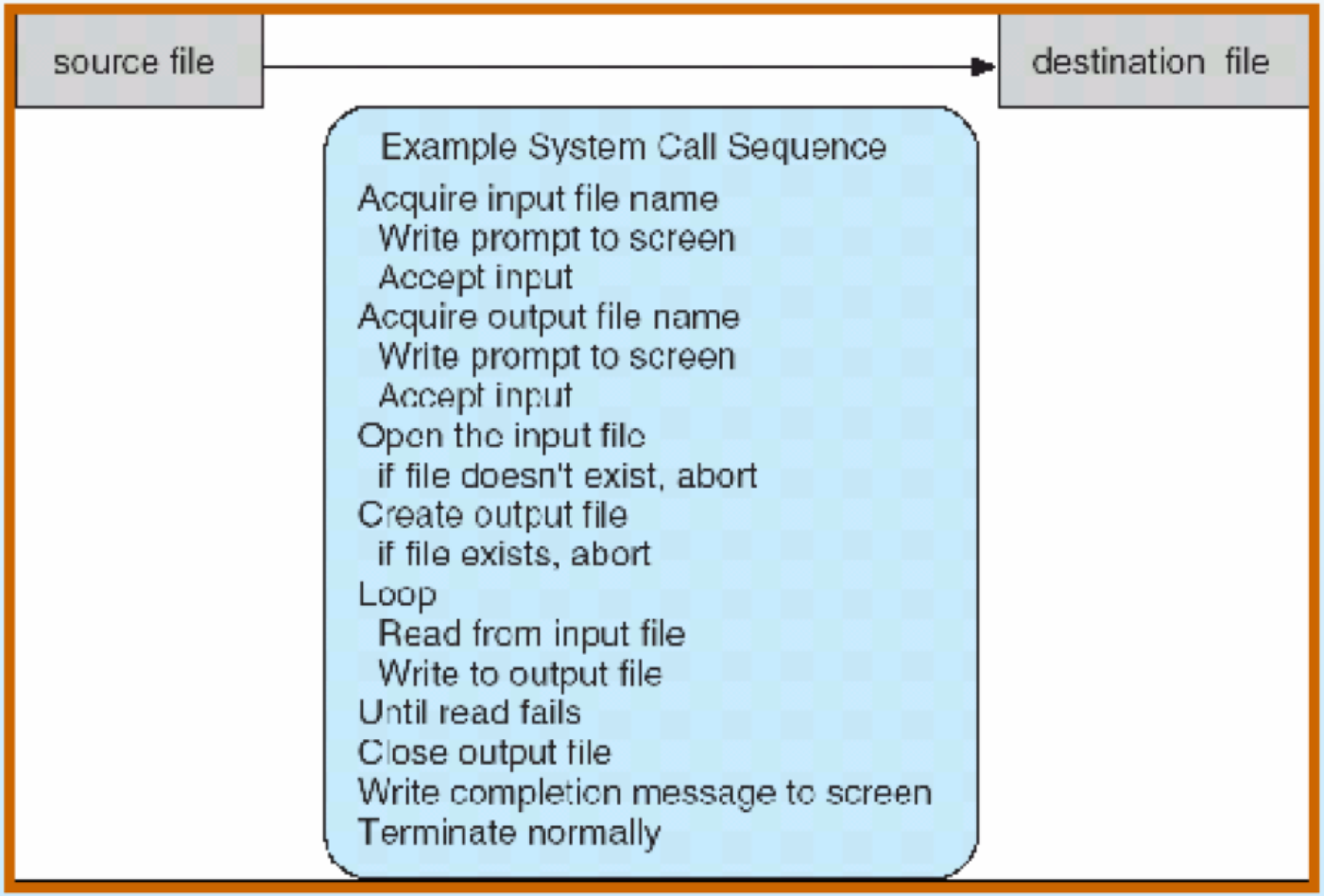
- Um processo pode resultar na execução de outros processos, chamados de processos filhos:
  - Características para a hierarquia de processos:
    - Comunicação (interação) e Sincronização;
    - Segurança e proteção;
    - Uma árvore de no máximo três níveis.
  - Escalonadores de processos - processo que escolhe qual será o processo a ser executado;
    - Diversas técnicas para escalonamento de processos.
  - Comunicação e sincronismo entre processos.

Processo - Chamada de Sistema

- Interface de programação fornecida pelo S.O.
- Normalmente escrita em linguagem de alto nível (C, C++, Java)
- Normalmente as aplicações utilizam uma *Application Program Interface* (API) que encapsula o acesso direto às *system calls*
- As APIs mais utilizadas são a Win32 API para Windows, a POSIX API para praticamente todas as versões UNIX e a JAVA API para a *Java Virtual Machine* (JVM)
- Motivos para utilizar APIs em vez das *System Calls* diretamente:
  - Portabilidade - Independência da plataforma
  - Esconder complexidade inerente às *system calls*
  - Acréscimo de funcionalidades que otimizam o desempenho
- O acesso às *system calls* está implementada em bibliotecas que são carregadas com as aplicações

Processo - Chamada de Sistema

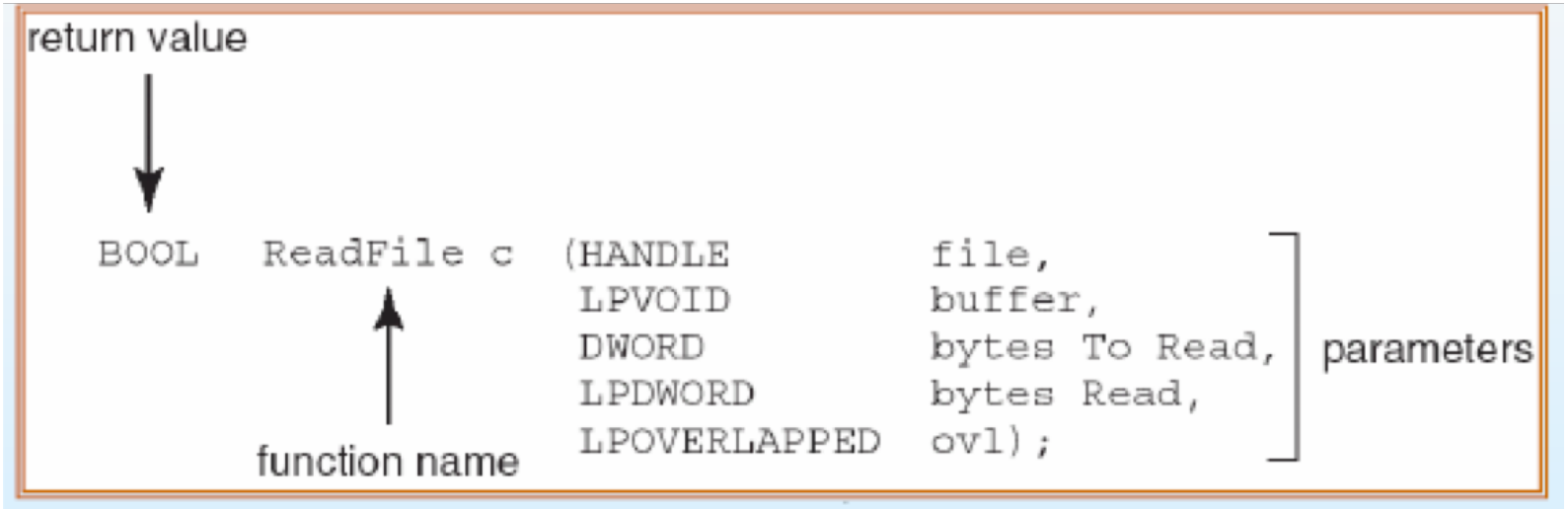
Sequência de *System Calls* para copiar o conteúdo de um arquivo para outro



Processo - Chamada de Sistema

A função $ReadFile()$ da Win32 API

- Uma função para ler o conteúdo de um arquivo



Processo - Chamada de Sistema

- Descrição dos parâmetros do $ReadFile()$
  - $HANDLEfile$ - the file to be read
  - $LPVOIDbuffer$ - a buffer where the data will be read into and written from
  - $DWORDbytesToRead$ - the number of bytes to be read into the buffer
  - $LPDWORDbytesRead$ - the number of bytes read during the last read
  - $LPOVERLAPPEDovl$ - indicates if overlapped I/O is being used

Processo - Chamada de Sistema

API Unix Standard

NAME

read - read from a file descriptor

SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>

int read(int fd, char *buf, size_t count);
```

DESCRIPTION

read reads up to *count* bytes from file descriptor *fd* into the buffer starting at *buf*.

RETURN VALUE

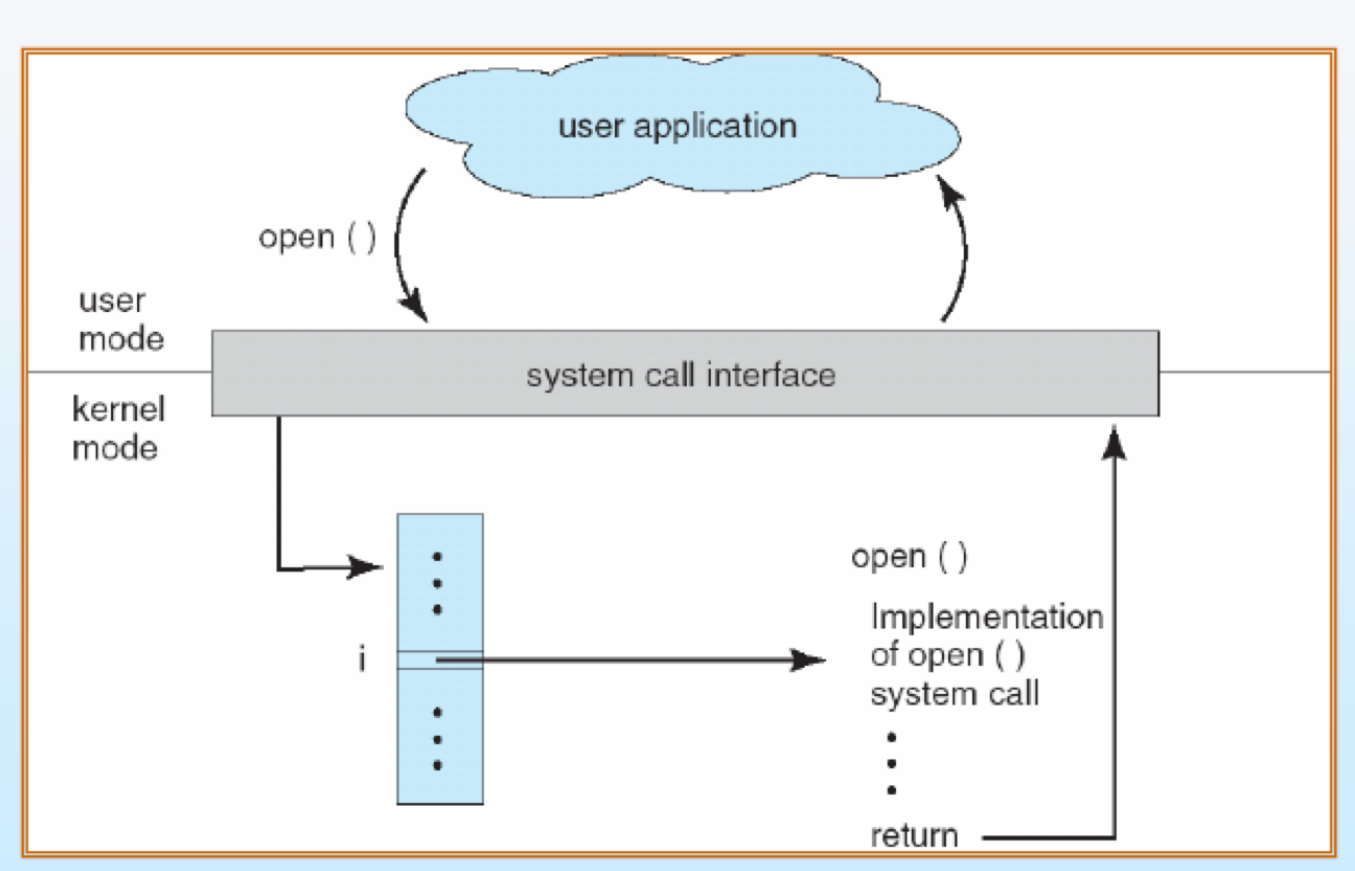
On success, the number of bytes read are returned (zero indicates end of file). On error, -1 is returned, and *errno* is set appropriately.

Processo - Chamada de Sistema

- A cada *system call* está associado um número
  - A interface mantém uma tabela com o endereço de cada *system call handler* que é indexada pelo número da *system call*
- Através deste tabela, o respectivo *handler* é invocado no *kernel*
  - Os parâmetros da *system call* são transferidos para o *kernel*
  - Uma vez executado, o resultado e os parâmetros de retorno são transferidos para o programa utilizador, como se tivesse havido uma invocação de uma função normal.
- A aplicação que invoca a *system call* não precisa saber como este é implementado
  - Só precisa obedecer à sintaxe da API (assinatura do método) e estar à espera dos resultados da invocação
  - Precisa conhecer o comportamento associado à *system call*
  - Os detalhes da interface do sistema são escondidos pela API (são geridos pela biblioteca *run-time* - camada de funções de biblioteca que são incluídas na aplicação quando da compilação e carregamento do arquivo executável.

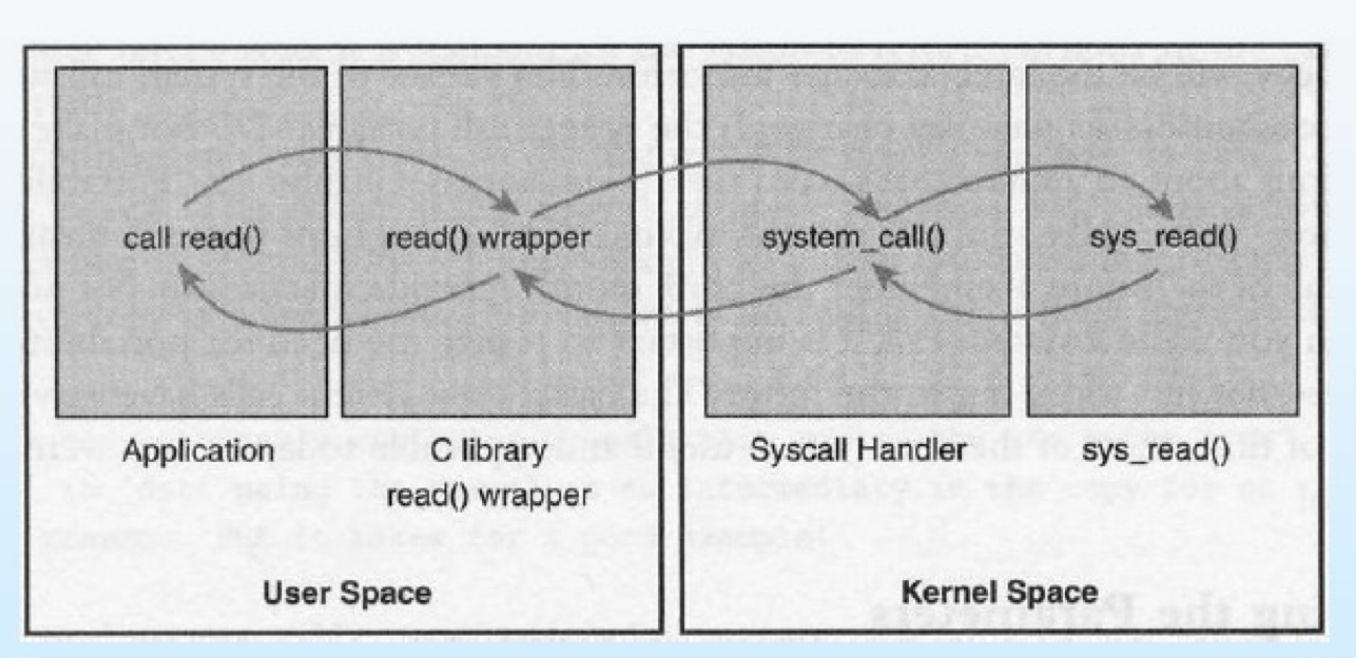
Processo - Chamada de Sistema

API - *System Call* - Relacionamento com o S.O.



Processo - Chamada de Sistema

Transição para as *Syscalls* em Linux



Processo - Chamada de Sistema

Linux System Calls Numbers

```
/*
 * This file contains the system call numbers.
 */

#define __NR_restart_syscall      0
#define __NR_exit                 1
#define __NR_fork                 2
#define __NR_read                 3
#define __NR_write                4
#define __NR_open                 5
#define __NR_close                6
#define __NR_waitpid              7
#define __NR_creat                8
#define __NR_link                 9
#define __NR_unlink              10
#define __NR_execve              11
#define __NR_chdir               12
#define __NR_time                13
#define __NR_mknod               14
#define __NR_chmod               15
#define __NR_lchown              16
#define __NR_break               17
#define __NR_oldstat             18
#define __NR_lseek               19
#define __NR_getpid              20
#define __NR_mount               21
#define __NR_umount              22
#define __NR_setuid              23
#define __NR_getuid              24
#define __NR_stime               25
#define __NR_ptrace              26
#define __NR_alarm               27
#define __NR_oldfstat            28
#define __NR_pause               29
#define __NR_utime               30
#define __NR_stty                31
#define __NR_gtty                32
#define __NR_access              33
#define __NR_nice                34
#define __NR_ftime               35
#define __NR_sync                36
#define __NR_kill                37
#define __NR_rename              38
#define __NR_mkdir               39
#define __NR_rmdir               40
#define __NR_dup                 41
#define __NR_pipe                42
```

/usr/src/linux/include/asm-i386/unistd.h

Processo - Chamada de Sistema

Linux Syscall Table

```
.data
ENTRY(sys_call_table)
        .long sys_restart_syscall       /* 0 - old "setup()" system call, used for restarting */
        .long sys_exit
        .long sys_fork
        .long sys_read
        .long sys_write
        .long sys_open          /* 5 */
        .long sys_close
        .long sys_waitpid
        .long sys_creat
        .long sys_link
        .long sys_unlink        /* 10 */
        .long sys_execve
        .long sys_chdir
        .long sys_time
        .long sys_mknod
        .long sys_chmod         /* 15 */
        .long sys_lchown16
        .long sys_ni_syscall    /* old break syscall holder */
        .long sys_stat
        .long sys_lseek
        .long sys_getpid        /* 20 */
        .long sys_mount
        .long sys_oldumount
        .long sys_setuid16
        .long sys_getuid16
        .long sys_stime         /* 25 */
        .long sys_ptrace
        .long sys_alarm
        .long sys_fstat
        .long sys_pause
        .long sys_utime         /* 30 */
        .long sys_ni_syscall    /* old stty syscall holder */
        .long sys_ni_syscall    /* old gtty syscall holder */
        .long sys_access
        .long sys_nice
        .long sys_ni_syscall    /* 35 - old ftime syscall holder */
        .long sys_sync
        .long sys_kill
        .long sys_rename
        .long sys_mkdir
        .long sys_rmdir         /* 40 */
        .long sys_dup
        .long sys_pipe
```

/usr/src/linux/arch/i386/kernel/entry.S

Processo - Chamada de Sistema

Invocação direta de *System Call*

- Programa em Assembler que invoca os system calls **write()** e **exit()** através da instrução int 0x80

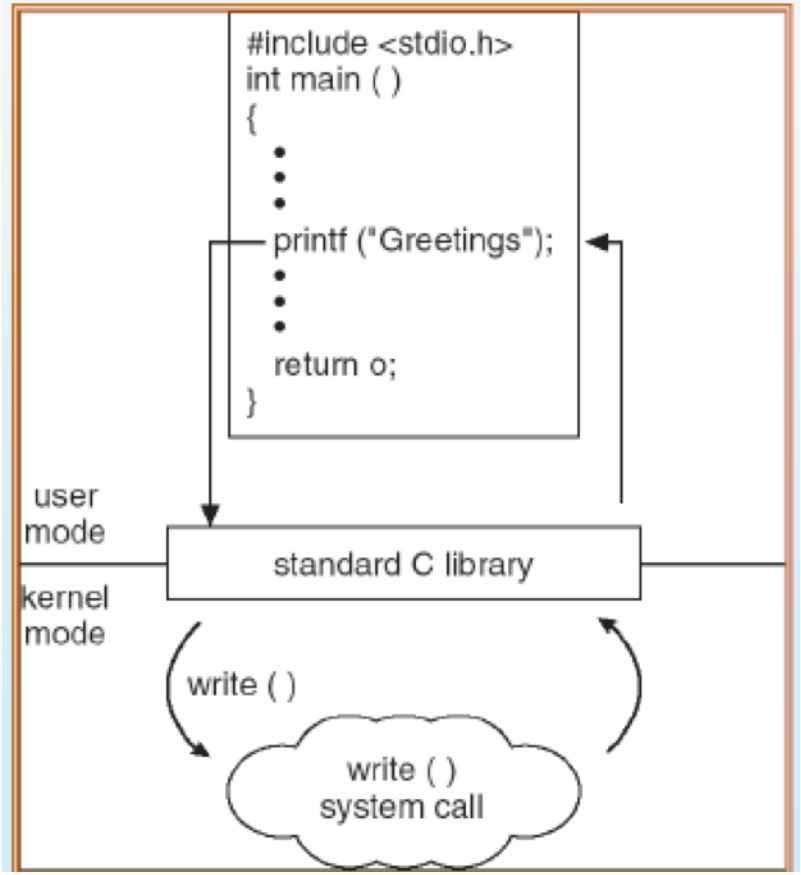
```
section .data                                ;section declaration

msg     db      "Hello, world!",0xa          ;our dear string
len     equ     $ - msg                      ;length of our dear string

section .text                                ;section declaration

                        ;we must export the entry point to the ELF linker or
    global _start       ;loader. They conventionally recognize _start as their
                        ;entry point. Use ld -e foo to override the default.

_start:

;write our string to stdout

        mov     edx,len ;third argument: message length
        mov     ecx,msg ;second argument: pointer to message to write
        mov     ebx,1   ;first argument: file handle (stdout)
        mov     eax,4   ;system call number (sys_write)
        int     0x80    ;call kernel

;and exit

        mov     ebx,0   ;first syscall argument: exit code
        mov     eax,1   ;system call number (sys_exit)
        int     0x80    ;call kernel
```

Processo - Chamada de Sistema

- Na arquitetura Intel a partir do Pentium II, duas novas instruções permitem a realização de *system calls* mais rapidamente
  - **sysentry** permite a entrada no sistema sem passar por uma interrupção de software
  - **sysexit** permite a saída do *kernel* pelo mesmo mecanismo
  - O *kernel linux* utiliza estas instruções preferencialmente a partir da versão 2.6
- A invocação destas instruções faz-se pela invocação direta de código *assembler* colocado pelo *kernel* em uma página específica de todos os processos (*virtual dynamic shared object - vdso*
  - O ponto de entrada é designado por $kernel_v syscall$
  - Endereço pode variar por distribuição e formato executável
- Ver referência `http://manugarg.googlepages.com/systemcallinlinux2_6.html`

Processo - Chamada de Sistema

Programa em C que invoca a função de biblioteca $printf()$, que por sua vez chama a *system call write()*



Processo - Chamada de Sistema

- Modos de acesso
  - Modo usuário;
  - Modo *Kernel* ou Supervisor ou Núcleo;
  - São determinados por um conjunto de bits localizados no registrador de status do processador PSW (*program status word*)
    - Por meio desse registrador o hardware verifica se a instrução pode ou não ser executada pela aplicação.
  - Protege o próprio *kernel* do Sistema Operacional na RAM contra acessos indevidos.

Processo - Chamada de Sistema

- Modo Usuário
  - Aplicações não têm acesso direto aos recursos da máquina, ou seja, ao hardware.
  - Quando o processador trabalha no modo usuário, a aplicação só pode executar **instruções sem privilégio, com um acesso reduzido de instruções**
  - Por que? Para garantir a **segurança** e a **integridade** do sistema.

Processo - Chamada de Sistema

- Modo *Kernel*
  - Aplicações têm acesso direto aos recursos da máquina, ou seja, ao hardware
  - **Operações com privilégios**
  - Quando o processador trabalha no modo *kernel*, a aplicação tem **acesso ao conjunto total de instruções**
  - Apenas o S.O. tem acesso às instruções privilegiadas.

Processo - Chamada de Sistema

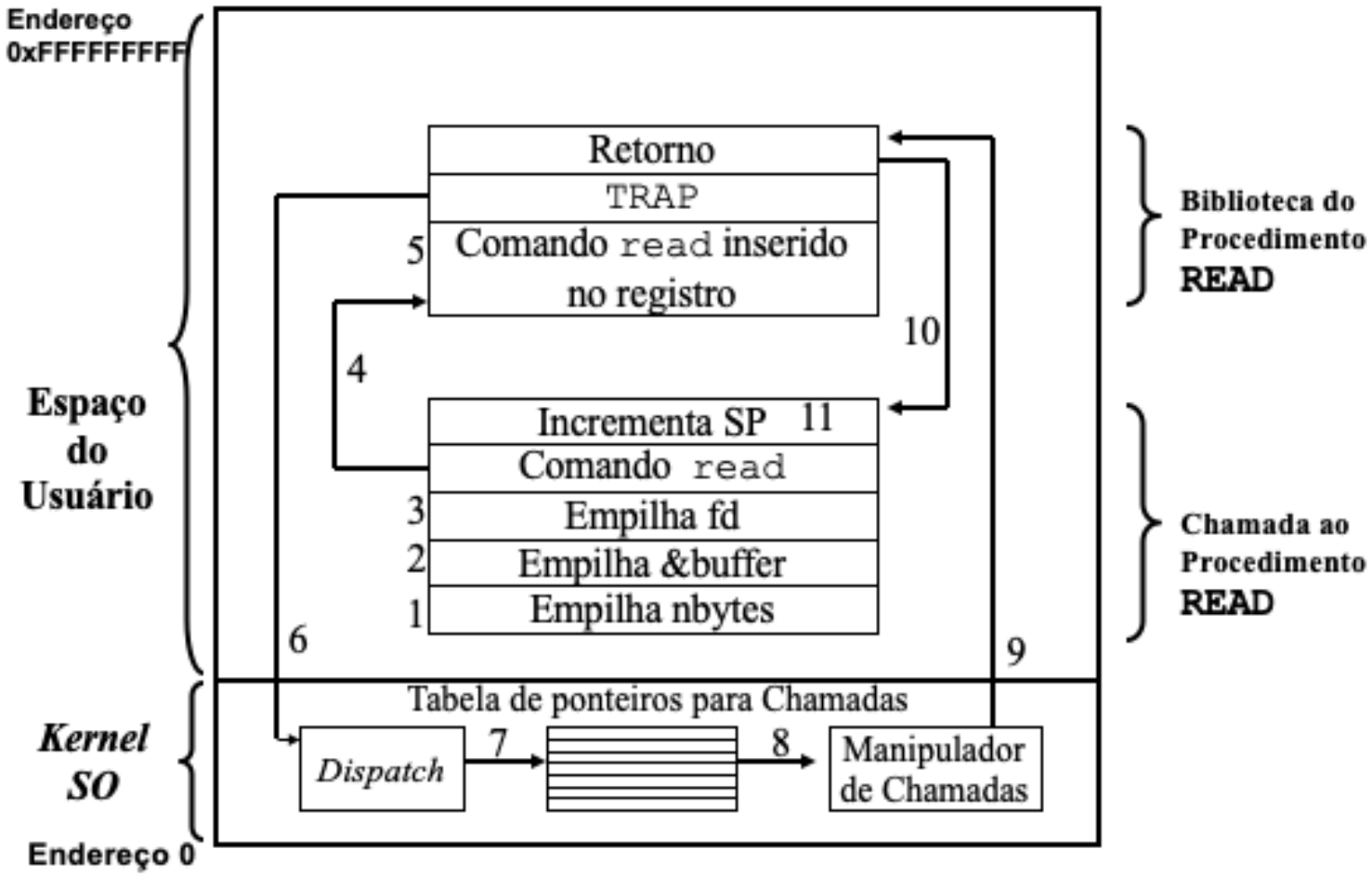
- Se uma aplicação precisa realizar alguma instrução privilegiada, ela realiza uma *chamada de sistema* que altera do modo usuário para o modo *kernel*
- Chamadas de sistemas são a **porta de entrada** para o modo *kernel*
  - São a interface entre os programas do usuário no modo usuário e o Sistema Operacional no modo *kernel*
  - As chamadas se diferem de S.O. para S.O., no entanto, os conceitos relacionados às chamadas de sistemas são similares independentemente de S.O.

Processo - Chamada de Sistema

- **TRAP**: instrução que permite acesso ao modo *kernel*
- Exemplo
  - Instrução no UNIX
  - $count = read(fd, buffer, nbytes);$

Tem-se: arquivo a ser lido, ponteiro para o *buffer* e bytes a serem lidos O programa sempre deve checar o retorno da chamada de sistema para saber se algum erro ocorreu!!!!

Sistema Operacional



Processo - Chamada de Sistema

- Exemplos de chamadas de sistema
  - Chamadas para gerenciamento de processos
    - Fork (createProcess - WIN32) - cria processo
  - Chamadas para gerenciamento de diretórios
    - Mount - monta um diretório
  - Chamadas para gerenciamento de arquivos
    - close (CloseHandle - WIN32) - fechar um arquivo
  - Outros tipos de chamadas
    - chmod - modifica permissões

Processo - Término

- Condições que podem provocar o término de um processo
  - Saída normal (voluntária)
  - Saída por erro (voluntária)
  - Erro fatal (involuntária)
  - Destruído por outro processo (involuntária)

Processo - Término

Condições

- Término normal (voluntária)
  - A tarefa a ser executada é finalizada
  - Chamadas: exit (UNIX) e ExitProcess (Windows)
- Término com erro (voluntária)
  - O processo sendo executado não pode ser finalizado: gcc filename.c, o arquivo filename.c não existe.

Processo - Término

Condições

- Término com erro fatal (involuntária)
  - Erro causado por algum erro no programa (*bug*)
    - Divisão por 0 (zero);
    - Referência à memória inexistente ou não pertencente ao processo;
    - Execução de uma instrução ilegal.
- Término causado por algum outro processo (involuntária)
  - kill (UNIX) e TerminateProcess (Windows)

Continuemos com os nossos .... PROCESSOS!!!!!!