Sistemas Operacionais

Prof. Dr. Helder Oliveira

Plano de Aula

- Componentes dos Sistemas Operacionais.
- Componentes uteis para os Sistemas Operacionais.
- Chamadas de sistema para gerenciamento de processos
- Chamadas de sistema para gerenciamento de arquivos
- Chamadas de sistema para gerenciamento de diretórios
- Chamadas de sistema diversas

Sistemas Operacionais

- Sistemas operacionais
  - Conceitos e abstrações básicos, como:
  - Processos, espaços de endereços e arquivos, que são fundamentais para compreendê-los.

Processo

- Um processo é basicamente um programa em execução.
- Associado a cada processo está o espaço de endereçamento, uma lista de posições de memória que vai de 0 a algum máximo, onde o processo pode ler e escrever.
- O espaço de endereçamento contém o programa executável, os dados do programa e sua pilha.
- Associado com cada processo há um conjunto de recursos, em geral abrangendo registradores, uma lista de arquivos abertos, alarmes pendentes, listas de processos relacionados e todas as demais informações necessárias para executar um programa.

Processo

- Um processo é na essência um contêiner que armazena todas as informações necessárias para executar um programa.
- A maneira mais fácil de compreender intuitivamente um processo é pensar a respeito do sistema de multiprogramação.

Compreendendo processo

- Imagine….
- Programa de edição de vídeo.
  - Converter um vídeo de uma hora para um determinado
  - Enquanto espera … Navegar na web.
  - Enquanto isso, um processo em segundo plano que desperta de tempos em tempos para conferir o e-mail que chega pode ter começado a ser executado.
  - Pelo menos três processos ativos:
    - Editor de vídeo
    - Navegador da web
    - Receptor de e-mail.
  - Periodicamente, o sistema operacional decide parar de executar um processo e começa a executar outro, talvez porque o primeiro utilizou mais do que sua parcela de tempo da CPU no último segundo ou dois.

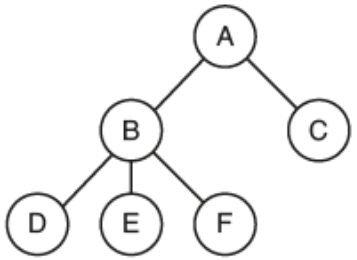
Compreendendo processo

- Quando um processo é suspenso temporariamente ele deve ser reiniciado mais tarde no exato mesmo estado em que estava quando foi parado.
- Informações do processo devem ser salvas durante a suspensão.
- O processo pode ter vários arquivos abertos para leitura ao mesmo tempo.
  - Há um ponteiro associado com cada um desses arquivos dando a posição atual.
  - Chamada read precisa ler os dados corretos após o processo ser reiniciado.
  - Tabela de processos guarda informações a respeito de cada processo.
- As principais chamadas de sistema de gerenciamento de processos são as que lidam com a criação e o término de processos.

Processos

- Um processo chamado de interpretador de comandos ou shell lê os comandos de um terminal.
- Um processo pode criar um ou mais processos (chamados de processos filhos), e estes por sua vez podem criar processos filhos.
- Comunicação entre processos é quando processos relacionados que estão cooperando para finalizar alguma tarefa muitas vezes precisam comunicar-se entre si e sincronizar as atividades.

FIGURA 1.13 Uma árvore de processo. O processo *A* criou dois processos filhos, *B* e *C*. O processo *B* criou três processos filhos, *D*, *E* e *F*.



Processos

- Outras chamadas de sistemas de processos permitem requisitar mais memória (ou liberar memória não utilizada), esperar que um processo filho termine e sobrepor seu programa por um diferente.
- Há ocasionalmente uma necessidade de se transmitir informação para um processo em execução que não está parado esperando por ela.
  - Ex:
    - Um processo que está se comunicando com outro em um computador diferente envia mensagens para o processo remoto por intermédio de uma rede de computadores.
    - Fazer uso de um temporizador.

Processos

- UID (User IDentification — identificação do usuário)
  - Todo processo iniciado tem a UID da pessoa que o iniciou.
  - Um processo filho tem a mesma UID que o seu processo pai.
- GID (Group IDentification — identificação do grupo)
  - Usuários podem ser membros de grupos, cada qual com uma GID.
- UID com poder especial, chamada de:
  - Superusuário (em UNIX)
  - Administrador (no Windows),
  - Pode passar por cima de muitas das regras de proteção.

Espaço de endereçamento

- Todo computador tem alguma memória principal que ele usa para armazenar programas em execução.
- Em sistemas operacionais muito simples, apenas um programa de cada vez está na memória.
- Em sistemas operacionais mais sofisticados múltiplos programas podem está na memória ao mesmo tempo.
  - Para evitar que interfiram entre si (e com o sistema operacional), algum tipo de mecanismo de proteção é necessário.
  - Embora esse mecanismo deva estar no hardware, ele é controlado pelo sistema operacional.

Espaço de endereçamento

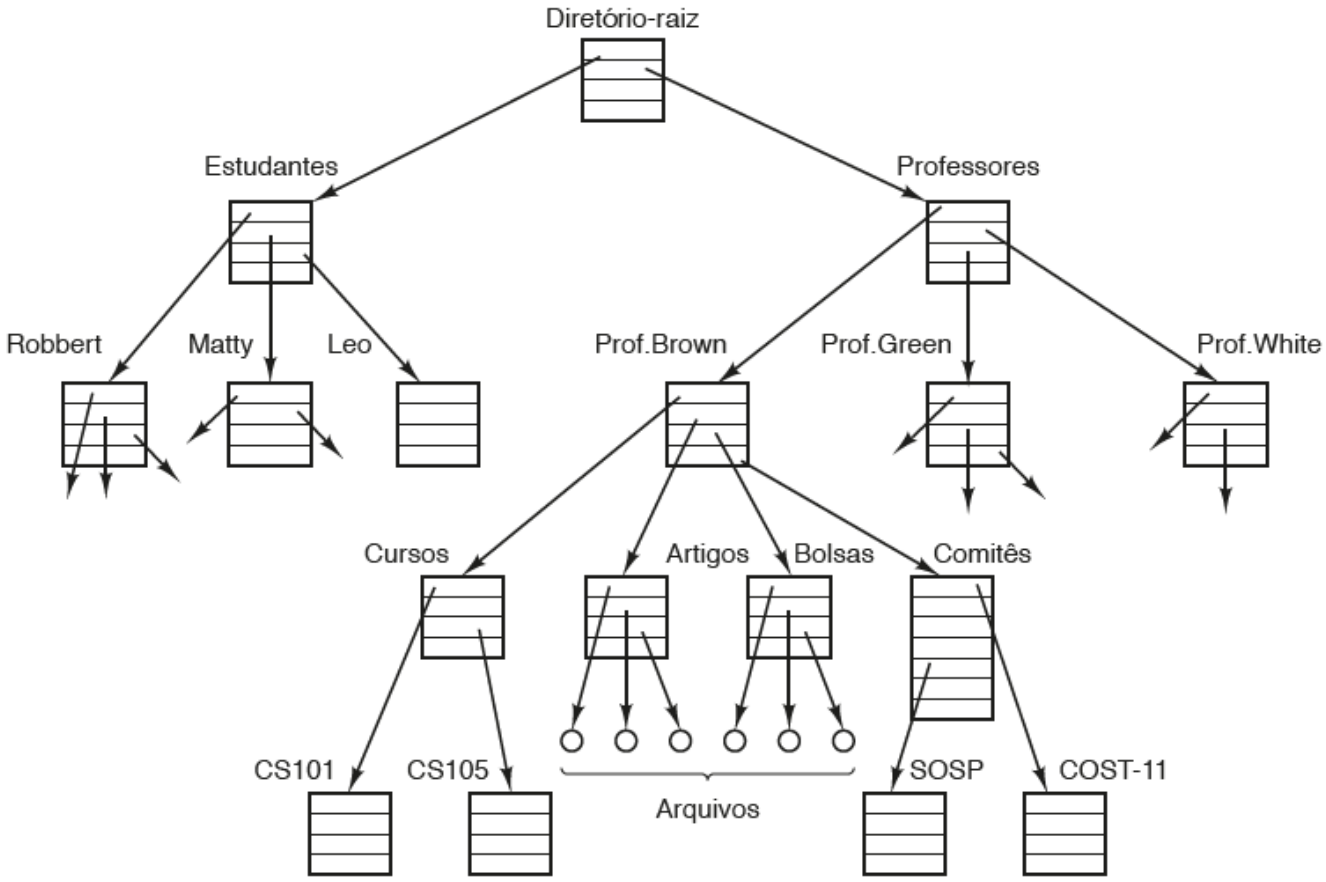
- Cada processo tem algum conjunto de endereços que ele pode usar, tipicamente indo de 0 até algum máximo.
- Em muitos computadores os endereços são de 32 ou 64 bits, dando um espaço de endereçamento de 2^{32} e 2^{64} .
- O que acontece se um processo tem mais espaço de endereçamento do que o computador tem de memória principal e o processo quer usá-lo inteiramente?
  - Hoje: memória virtual
  - O sistema operacional mantém parte do espaço de endereçamento na memória principal e parte no disco.

Arquivos

- Sistemas de Arquivos – Fundamental.
- Chamadas de sistema são obviamente necessárias para criar, remover, ler e escrever arquivos.
- Antes que um arquivo possa ser lido, ele deve:
  - Ser localizado no disco e aberto, e após ter sido lido, deve ser fechado.
- Diretório: maneira de agrupar os arquivos.
  - Chamadas de sistema usadas para:
    - Criar e remover diretórios.
    - Colocar um arquivo existente em um diretório.
    - Remover um arquivo de um diretório.
  - Entradas de diretório podem ser de arquivos ou de outros diretórios.

Arquivos

FIGURA 1.14 Um sistema de arquivos para um departamento universitário.



Arquivos

- Ambas as hierarquias de processos e arquivos são organizadas como árvores, mas a similaridade para aí.
- Hierarquias de processos em geral não são muito profundas (mais do que três níveis é incomum), enquanto hierarquias de arquivos costumam ter quatro, cinco, ou mesmo mais níveis de profundidade.
- Hierarquias de processos tipicamente têm vida curta, em geral minutos no máximo, enquanto hierarquias de diretórios podem existir por anos.
- Propriedade e proteção também diferem para processos e arquivos. Normalmente, apenas um processo pai pode controlar ou mesmo acessar um processo filho, mas quase sempre existem mecanismos para permitir que arquivos e diretórios sejam lidos por um grupo mais amplo do que apenas o proprietário.

Arquivos

- Todo arquivo dentro de uma hierarquia de diretório pode ser especificado fornecendo o seu nome de caminho a partir do topo da hierarquia do diretório, o diretório-raiz.
- Ex:
  - /Professores/Prof.Brown/Cursos/CS101
    - A primeira barra indica que o caminho é absoluto
    - Se fosse no windows \Professores\Prof.Brown\Cursos\ CS101 (razões históricas)

Arquivos

- Lidando com mídia removível no UNIX
  - Agregado à árvore principal
  - Chamada mount

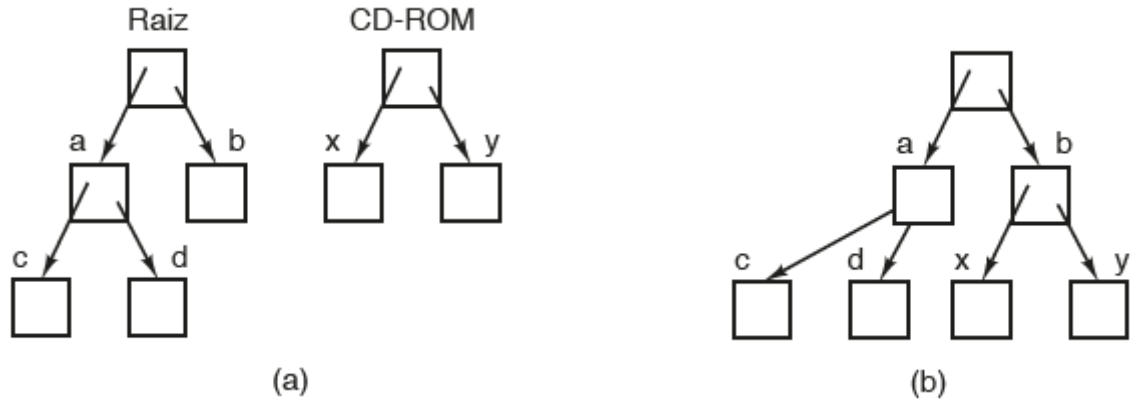


FIGURA 1.15 (a) Antes da montagem, os arquivos no CD-ROM não estão acessíveis. (b) Depois da montagem, eles fazem parte da hierarquia de arquivos.

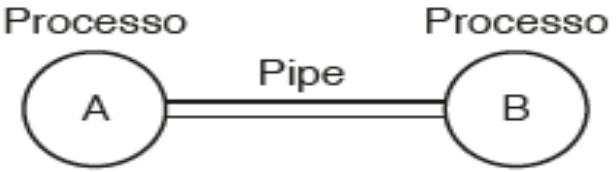
Arquivos

- Arquivo especial (UNIX)
  - Permitem que dispositivos de E/S se pareçam com arquivos.
  - Podem ser lidos e escritos com as mesmas chamadas de sistema que são usadas para ler e escrever arquivos.
- Existem dois tipos especiais:
  - Arquivos especiais de bloco
    - são usados para modelar dispositivos que consistem em uma coleção de blocos aleatoriamente endereçáveis, como discos.
  - Arquivos especiais de caracteres
    - são usados para modelar impressoras, modems e outros dispositivos que aceitam ou enviam um fluxo de caracteres.
- Arquivos especiais são mantidos no diretório /dev.
  - Ex: /dev/lp pode ser a impressora

Arquivos

- **Pipe**
  - Uma espécie de pseudoarquivo que pode ser usado para conectar dois processos.
  - Se os processos A e B querem conversar usando um pipe, eles têm de configurá-lo antes.
  - Quando o processo A quer enviar dados para o processo B, ele escreve no pipe como se ele fosse um arquivo de saída.
  - O processo B pode ler os dados a partir do pipe como se ele fosse um arquivo de entrada.
  - A comunicação entre os processos em UNIX se parece muito com a leitura e escrita de arquivos comuns.

FIGURA 1.16 Dois processos conectados por um pipe.



Entrada e Saída

- Todos os computadores têm dispositivos físicos para obter entradas e produzir saídas.
- Existem muitos tipos de dispositivos de entrada e de saída.
  - Ex:
    - Teclados, monitores, impressoras e assim por diante.
  - Cabe ao sistema operacional gerenciá-los.
- Todo sistema operacional tem um subsistema de E/S para gerenciar os dispositivos de E/S.

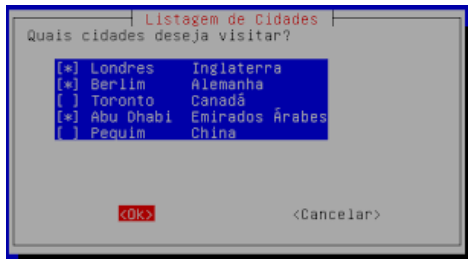
Proteção

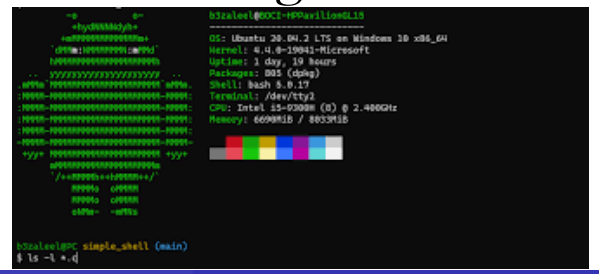
- Computadores contêm grandes quantidades de informações que os usuários muitas vezes querem proteger e manter confidenciais.
- Sistema operacional gerencia a segurança e garante que arquivos sejam acessíveis somente por usuários autorizados.
- Arquivos em UNIX são protegidos designando--se a cada arquivo um código de proteção binário de 9 bits.
  - O código de proteção consiste de três campos de 3 bits.
  - Ex: **rwx r-x —x**
  - O proprietário, outros membros do grupo, todos os demais

--- => nenhuma permissão;
r-- => permissão de leitura;
r-x => leitura e execução;
rw- => leitura e gravação;
rwx => leitura, gravação e execução.

Componentes importantes e úteis para o S.O

- **O interpretador de comandos (shell)**
  - Faz um uso intensivo de muitos aspectos do sistema operacional.
  - O sistema operacional é o código que executa as chamadas de sistema.
  - Editores, compiladores, montadores, ligadores (linkers), programas utilitários e interpretadores de comandos definitivamente não fazem parte do sistema operacional, mesmo que sejam importantes e úteis.
  - A principal interface entre um usuário sentado no seu terminal e o sistema operacional – (pode usar a interface de usuário gráfica).
  - Muitos shells existem, incluindo, sh, csh, ksh e bash.
  - A maioria dos computadores pessoais usa hoje uma interface gráfica GUI.





Componentes importantes e úteis para o S.O

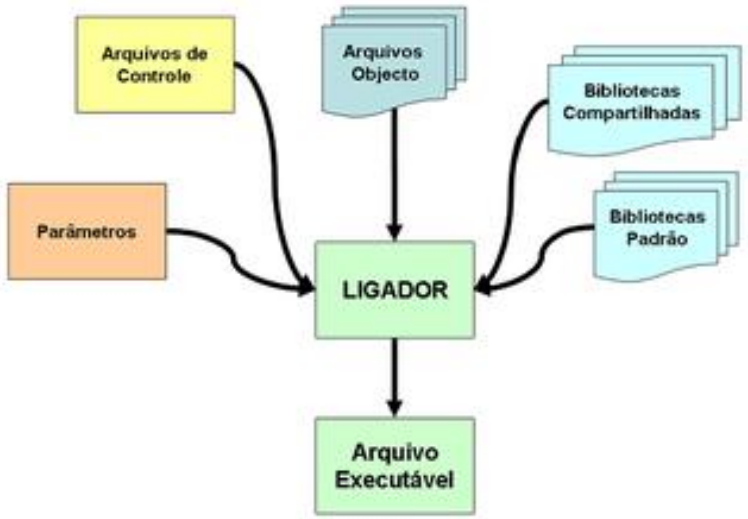
- **Tradutor**
  - Montador
    - É o utilitário responsável por traduzir um programa-fonte em linguagem de montagem para um programa em linguagem de máquina.
    - A linguagem de montagem é particular para cada processador, assim como a linguagem de máquina.
    - Programas assembly não podem ser portados entre máquinas diferentes.
    - Ex: Pascal, Fortran, C, etc
  - Compilador
    - É o utilitário responsável por gerar (de um programa escrito em linguagem de alto nível) um programa de linguagem de máquina não executável.
    - As linguagens de alto nível não tem nenhuma relação direta com a máquina (fica a cargo do compilador).
    - Programas-fonte podem ser portados entre computadores de diversos fabricantes.
    - Converte o programa completo para depois ser executado.

Componentes importantes e úteis para o S.O

- **Interpretador**
  - É considerado um tradutor que não gera módulo-objeto;
  - Durante a execução de um programa, traduz cada instrução e a executa imediatamente;
  - São mais lentos que os compilados.
  - DESVANTAGEM:
    - Tempo gasto na tradução das instruções de um programa;
  - VANTAGEM:
    - Permitir a implementação de tipos de dados dinâmicos;

Componentes importantes e úteis para o S.O

- **Ligadores (linkers)**
  - É o utilitário responsável por gerar, a partir de um ou mais módulos-objeto, um único programa executável.
  - Resolve todas as referências simbólicas entre os módulos;
  - Reserva memória para a execução do programa;

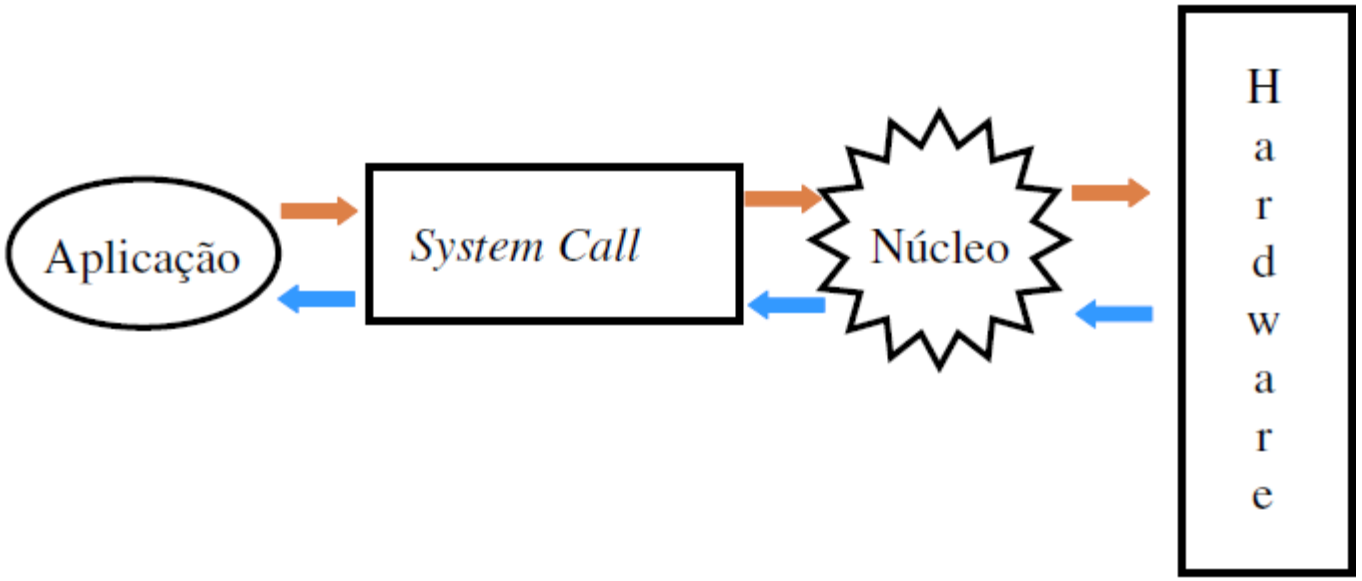


Componentes importantes e úteis para o S.O

- **Loader**
  - É o utilitário responsável por carregar na memória principal um programa para ser executado;
  - Pode ser: Absoluto ou Relocável;
  - ABSOLUTO: Só necessita conhecer o endereço de memória inicial e o tamanho do módulo para realizar o carregamento;
  - RELOCÁVEL: O Programa é carregado em qualquer posição da memória (loader fica responsável pela relocação no momento do carregamento);

Chamadas de sistema

- O usuário (ou aplicação), quando deseja solicitar algum serviço do sistema, realiza uma chamada a uma de suas rotinas (ou serviços) através da system calls (chamadas ao sistema).



Chamadas de sistema

- SO roda em Modo kernel, supervisor ou núcleo
  - Protege o hardware da ação direta do usuário.
- Os demais programas rodam em modo usuário e fazem chamadas ao kernel para terem acesso aos dispositivos.

Chamadas de sistema

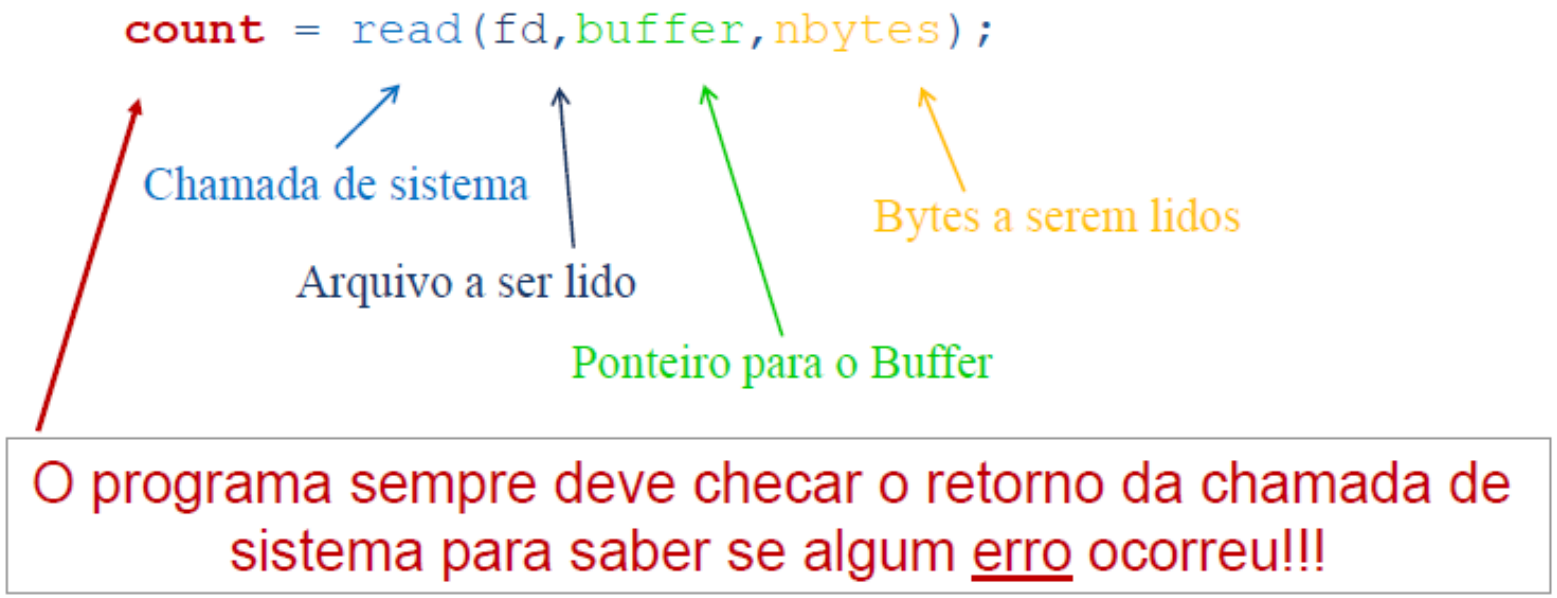
- Sistemas operacionais têm duas funções principais:
  - Fornecer abstrações para os programas de usuários e gerenciar os recursos do computador
  - A interação entre programas de usuários e o sistema operacional lida com o fornecimento de abstrações para os programas de usuários.
  - Ex:
    - Criar, escrever, ler e deletar arquivos.
  - A interface entre os programas de usuários e o sistema operacional diz respeito fundamentalmente a abstrações.
  - As chamadas de sistema disponíveis na interface variam de um sistema para outro.
    - Embora os conceitos subjacentes tendam a ser similares.

Chamadas de sistema

- Chamadas de sistema disponíveis na interface variam.
  - Estudaremos um sistema específico "**UNIX**"
  - Possui uma chamada de sistema read com três parâmetros:
    - Um para especificar o arquivo.
    - Um para dizer onde os dados devem ser colocados.
    - Um para dizer quantos bytes devem ser lidos.
- Lembre: Qualquer computador de uma única CPU pode executar apenas uma instrução de cada vez.
  - Se um processo estiver executando um programa de usuário em modo de usuário e precisa de um serviço de sistema, como ler dados de um arquivo, ele tem de executar uma instrução de armadilha (trap) para transferir o controle para o sistema operacional.
  - O sistema operacional verifica os parâmetros e descobre o que o processo que está chamando quer. Então ele executa a chamada de sistema e retorna o controle para a instrução seguinte à chamada de sistema.

Chamadas de sistema

- Chamada de sistema read.
  - Como quase todas as chamadas de sistema, ele é invocado de programas C chamando uma rotina de biblioteca com o mesmo nome que a chamada de sistema: read.



Chamadas de sistema

- **Instrução TRAP**
  - Instrução que permite o acesso ao modo kernel /núcleo.
  - Relativamente similar à instrução de chamada de rotina no sentido de que a instrução que a segue é tirada de um local distante e o endereço de retorno é salvo na pilha para ser usado depois.
  - Difere da instrução de chamada de rotina de duas maneiras fundamentais:
    - Primeiro: ela troca para o modo núcleo.
      - A instrução de chamada de rotina não muda o modo.
    - Segundo: em vez de dar um endereço relativo ou absoluto onde a rotina está localizada, a instrução TRAP não pode saltar para um endereço arbitrário.
      - Salta para um único local fixo,

Chamadas de sistema

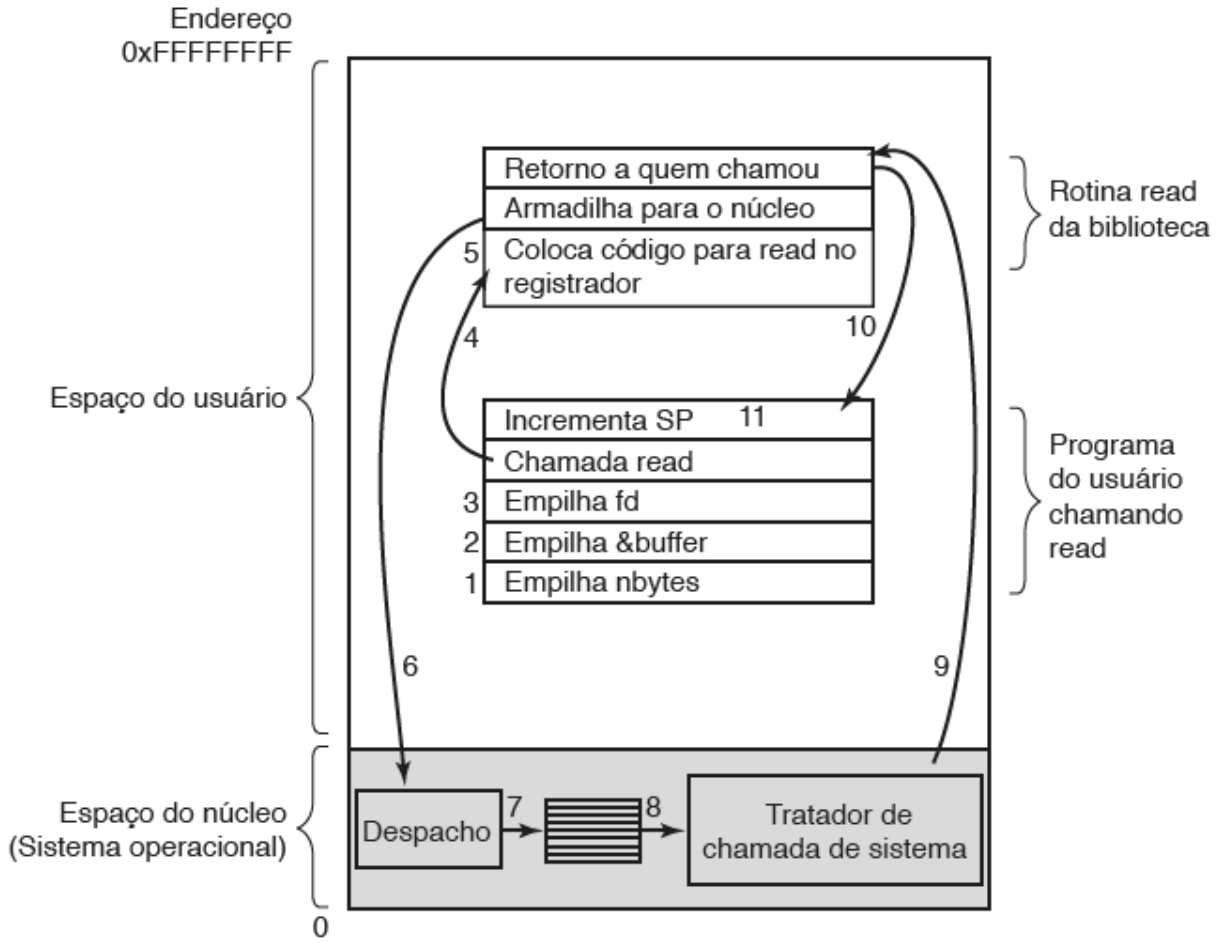


FIGURA 1.17 Os 11 passos na realização da chamada de sistema read (fd, buffer, nbytes).

- Examinando a chamada read
- O programa armazena os parâmetros na pilha (1 a 3);
- Chama a rotina *read* (4);
- A rotina de biblioteca coloca a chamada de sistema em um registrador (5);
- Executa um TRAP para passar do modo usuário para o núcleo (6);
- Despacha para a rotina correta de tratamento dessa chamada (7);
- Rotina de tratamento da chamada é executada (8);
- Controle retorna para a biblioteca no espaço do usuário, seguinte ao TRAP (9);
- Retorna a rotina ao programa do usuário (10);
- O programa do usuário deve limpar a pilha (11);

Chamadas de sistema

- **POSIX**
  - Define uma interface minimalista de chamadas de sistema à qual os sistemas UNIX em conformidade devem dar suporte.
  - Será utilizado para exemplificar chamadas simples no sistema.

Chamadas de sistema para gerenciamento de processos

- Exemplos de chamadas para gerenciamento de processos.
- Chamadas que lidam com o gerenciamento de processo.

FIGURA 1.18 Algumas das principais chamadas de sistema POSIX. O código de retorno *s* é –1 se um erro tiver ocorrido. Os códigos de retorno são os seguintes: *pid* é um processo id, *fd* é um descritor de arquivo, *n* é um contador de bytes, *position* é um deslocamento no interior do arquivo e *seconds* é o tempo decorrido. Os parâmetros são explicados no texto.

Gerenciamento de processos

Chamada	Descrição
pid = fork( )	Cria um processo filho idêntico ao pai
pid = waitpid(pid, &statloc, options)	Espera que um processo filho seja concluído
s = execve(name, argv, environp)	Substitui a imagem do núcleo de um processo
exit(status)	Conclui a execução do processo e devolve status

Chamadas de sistema para gerenciamento de processos

- A chamada **fork** é a única maneira para se criar um processo novo.
  - cria uma cópia exata do processo original, incluindo todos os descritores de arquivos, registradores — tudo.
  - Após a fork o processo original e a cópia seguem seus próprios caminhos separados.
  - Mudanças subsequentes em um deles não afetam o outro.
  - Retorna um valor, que é zero no processo filho e igual ao PID (Process IDentifier — identificador de processo) do processo filho no processo pai.
  - Usando o PID retornado, os dois processos podem ver qual é o processo pai e qual é o filho.

Chamadas de sistema para gerenciamento de processos

- Chamada chamada de sistema **waitpid**
  - Considere o caso do shell. Ele lê um comando do terminal, cria um processo filho, espera que ele execute o comando e então lê o próximo comando quando o processo filho termina.
  - Para esperar que o processo filho termine, o processo pai executa uma chamada de sistema waitpid.
  - Waitpid pode esperar por um processo filho específico ou por qualquer filho mais velho configurando-se o primeiro parâmetro em −1.
  - Quando waitpid termina, o endereço apontado pelo segundo parâmetro, statloc, será configurado como estado de saída do processo filho.

Chamadas de sistema para gerenciamento de processos

- Chamada de sistema **execve**
  - Quando um comando é digitado, o shell cria um novo processo.
  - Esse processo filho tem de executar o comando de usuário.
  - Ele o faz usando a chamada de sistema execve, que faz que toda a sua imagem de núcleo seja substituída pelo arquivo nomeado no seu primeiro parâmetro.

Chamadas de sistema para gerenciamento de processos

FIGURA 1.19 Um interpretador de comandos simplificado. Neste livro, presume-se que *TRUE* seja definido como 1.*

```
#define TRUE 1

while (TRUE) {                                  /* repita para sempre */
        type_prompt( );                         /* mostra prompt na tela */
        read_command(command, parameters);      /* le entrada do terminal */

        if (fork( ) != 0) {                     /* cria processo filho */
             /* Codigo do processo pai. */
             waitpid(–1, &status, 0);           /* aguarda o processo filho acabar */
        } else {
             /* Codigo do processo filho. */
             execve(command, parameters, 0);    /* executa o comando */
        }
}
```

Chamadas de sistema para gerenciamento de arquivos

- Chamadas que operam sobre arquivos individuais.

FIGURA 1.18 Algumas das principais chamadas de sistema POSIX. O código de retorno *s* é –1 se um erro tiver ocorrido. Os códigos de retorno são os seguintes: *pid* é um processo id, *fd* é um descritor de arquivo, *n* é um contador de bytes, *position* é um deslocamento no interior do arquivo e *seconds* é o tempo decorrido. Os parâmetros são explicados no texto.

Gerenciamento de arquivos

Chamada	Descrição
fd = open(file, how, ...)	Abre um arquivo para leitura, escrita ou ambos
s = close(fd)	Fecha um arquivo aberto
n = read(fd, buffer, nbytes)	Lê dados a partir de um arquivo em um buffer
n = write(fd, buffer, nbytes)	Escreve dados a partir de um buffer em um arquivo
position = lseek(fd, offset, whence)	Move o ponteiro do arquivo
s = stat(name, &buf)	Obtém informações sobre um arquivo

Chamadas de sistema para gerenciamento de arquivos

- Chamada de sistema **open**
  - Para ler ou escrever um arquivo, é preciso primeiro abri-lo.
  - Essa chamada especifica o nome do arquivo a ser aberto.
  - Pode ser um nome de caminho absoluto ou relativo ao diretório de trabalho
    - Como um código de O_RDONLY, O_WRONLY, ou O_RDWR, significando aberto para leitura, escrita ou ambos.
  - Para criar um novo arquivo, o parâmetro O_CREAT é usado.
  - O descritor de arquivos retornado pode então ser usado para leitura ou escrita.
- Chamada de sistema **close**
  - Fecha o arquivo aberto
  - O descritor disponível pode ser reutilizado em um open subsequente.

Chamadas de sistema para gerenciamento de arquivos

- Chamada de sistema **read e write**
  - Embora a maioria dos programas leia e escreva arquivos sequencialmente, alguns programas de aplicativos precisam ser capazes de acessar qualquer parte de um arquivo de modo aleatório.
  - Associado a cada arquivo há um ponteiro que indica a posição atual no arquivo.
- Chamada de sistema **lseek**
  - Muda o valor do ponteiro de posição, de maneira que chamadas subsequentes para ler ou escrever podem começar em qualquer parte no arquivo.
  - Três parâmetros:
    - Primeiro é o descritor de arquivo para o arquivo.
    - Segundo é uma posição do arquivo.
    - Terceiro diz se a posição do arquivo é relativa ao começo, à posição atual ou ao fim do arquivo.

Chamadas de sistema para gerenciamento de diretórios

- Relacionam mais aos diretórios ou o sistema de arquivos como um todo, em vez de apenas um arquivo específico.

FIGURA 1.18 Algumas das principais chamadas de sistema POSIX. O código de retorno *s* é –1 se um erro tiver ocorrido. Os códigos de retorno são os seguintes: *pid* é um processo id, *fd* é um descritor de arquivo, *n* é um contador de bytes, *position* é um deslocamento no interior do arquivo e *seconds* é o tempo decorrido. Os parâmetros são explicados no texto.

Gerenciamento do sistema de diretório e arquivo

Chamada	Descrição
s = mkdir(name, mode)	Cria um novo diretório
s = rmdir(name)	Remove um diretório vazio
s = link(name1, name2)	Cria uma nova entrada, name2, apontando para name1
s = unlink(name)	Remove uma entrada de diretório
s = mount(special, name, flag)	Monta um sistema de arquivos
s = umount(special)	Desmonta um sistema de arquivos

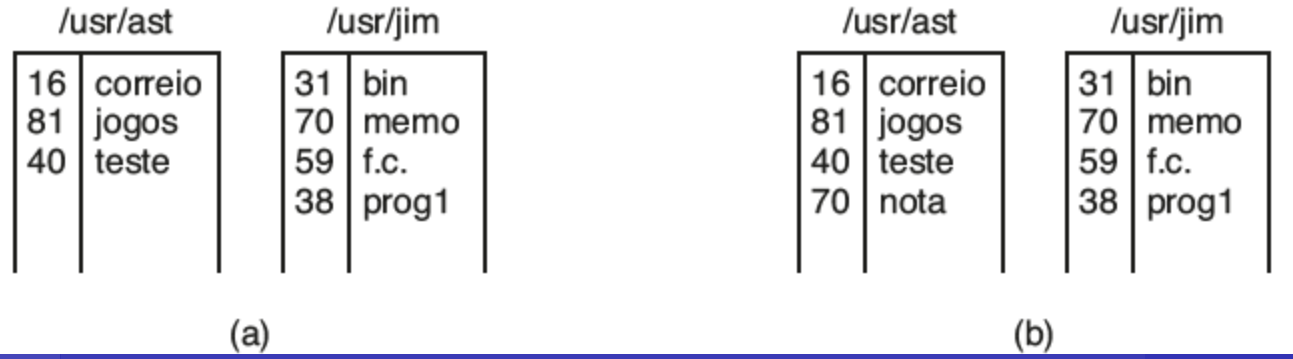
Chamadas de sistema para gerenciamento de diretórios

- Chamadas **mkdir** e **rmdir**
  - Criam e removem diretórios vazios
- Chamada **link**
  - Permite que o mesmo arquivo apareça sob dois ou mais nomes, muitas vezes em diretórios diferentes.
  - Uso típico é permitir que vários membros da mesma equipe de programação compartilhem um arquivo comum.
    - Cada um deles tendo o arquivo aparecendo no seu próprio diretório, possivelmente sob nomes diferentes.
  - Compartilhar um arquivo não é o mesmo que dar a cada membro da equipe uma cópia particular;
  - Ter um arquivo compartilhado significa que as mudanças feitas por qualquer membro da equipe são instantaneamente visíveis para os outros membros, mas há apenas um arquivo.

- Exemplo de chamada **link**
  - Dois usuários, ast e jim, cada um com o seu próprio diretório com alguns arquivos.
  - Se ast agora executa um programa contendo a chamada de sistema

    link("/usr/jim/memo", "/usr/ast/nota");
  - O arquivo memo no diretório de jim estará aparecendo agora no diretório de ast sob o nome nota.
  - Todo arquivo UNIX tem um número único, o seu i-número, que o identifica.

FIGURA 1.21 (a) Dois diretórios antes da ligação de */usr/jim/memo* ao diretório *ast*. (b) Os mesmos diretórios depois dessa ligação.



Chamadas de sistema para gerenciamento de arquivos

- Chamada de sistema **mount**
  - Permite que dois sistemas de arquivos sejam fundidos em um.
  - Ao executar a chamada de sistema mount, o sistema de arquivos USB pode ser anexado ao sistema de arquivos-raiz.
  - Comando em C.

mount("/dev/sdb0", "/mnt", 0);

    - Primeiro parâmetro é o nome de um arquivo especial de blocos para a unidade de disco 0.
    - Segundo é o lugar na árvore onde ele deve ser montado.
    - Terceiro diz se o sistema de arquivos deve ser montado como leitura e escrita ou somente leitura.

Chamadas de sistema para gerenciamento de arquivos

- Chamada de sistema **mount**

FIGURA 1.22 (a) O sistema de arquivos antes da montagem. (b) O sistema de arquivos após a montagem.

Chamadas de sistema diversas

- Existe uma variedade de outras chamadas de sistema.
- Mostraremos algumas

FIGURA 1.18 Algumas das principais chamadas de sistema POSIX. O código de retorno *s* é –1 se um erro tiver ocorrido. Os códigos de retorno são os seguintes: *pid* é um processo id, *fd* é um descritor de arquivo, *n* é um contador de bytes, *position* é um deslocamento no interior do arquivo e *seconds* é o tempo decorrido. Os parâmetros são explicados no texto.

Diversas

Chamada	Descrição
s = chdir(dirname)	Altera o diretório de trabalho
s = chmod(name, mode)	Altera os bits de proteção de um arquivo
s = kill(pid, signal)	Envia um sinal para um processo
seconds = time(&seconds)	Obtém o tempo decorrido desde 1º de janeiro de 1970

Chamadas de sistema diversas

- Chamada de sistema **chdir**
  - Muda o diretório de trabalho atual.
  - Após uma chamada chdir("/usr/ast/test"); uma abertura no arquivo xyz abrirá /usr/ast/test/xyz.
  - O conceito de um diretório de trabalho elimina a necessidade de digitar (longos) nomes de caminhos absolutos a toda hora.

Chamadas de sistema diversas

- Chamada de sistema **chmod**
  - Torna possível mudar o modo de um arquivo.
  - Por exemplo, para tornar um arquivo como somente de leitura para todos, exceto o proprietário, poderia ser executado: chmod("file", 0644);
- Chamada de sistema **kill**
  - Maneira pela qual os usuários e os processos de usuários enviam sinais.
  - Se um processo está preparado para capturar um sinal em particular, então, quando ele chega, uma rotina de tratamento desse sinal é executada. Se o processo não está preparado para lidar com um sinal, então sua chegada mata o processo (daí seu nome).

API Win32 do Windows

- Microsoft definiu um conjunto de rotinas chamadas de API Win32 (Application Programming Interface) para acessar os serviços do sistema operacional.
- O Windows e o UNIX diferem de uma maneira fundamental em seus respectivos modelos de programação.
- Um programa UNIX consiste de um código que faz uma coisa ou outra, fazendo chamadas de sistema para ter determinados serviços realizados.
- Um programa Windows é normalmente direcionado por eventos.
  - O programa principal espera por algum evento acontecer, então chama uma rotina para lidar com ele.
  - Eventos típicos são teclas sendo pressionadas, o mouse sendo movido, um botão do mouse acionado, ou um disco flexível inserido.
  - Tratadores são então chamados para processar o evento, atualizar a tela e o estado do programa interno.

API Win32 do Windows

FIGURA 1.23 As chamadas da API Win32 que correspondem aproximadamente às chamadas UNIX da Figura 1.18. Vale a pena enfatizar que o Windows tem um número muito grande de outras chamadas de sistema, a maioria das quais não corresponde a nada no UNIX.

UNIX	Win32	Descrição
fork	CreateProcess	Cria um novo processo
waitpid	WaitForSingleObject	Pode esperar que um processo termine
execve	(nenhuma)	CreateProcess = fork + execve
exit	ExitProcess	Conclui a execução
open	CreateFile	Cria um arquivo ou abre um arquivo existente
close	CloseHandle	Fecha um arquivo
read	ReadFile	Lê dados a partir de um arquivo
write	WriteFile	Escreve dados em um arquivo
lseek	SetFilePointer	Move o ponteiro do arquivo
stat	GetFileAttributesEx	Obtém vários atributos do arquivo
mkdir	CreateDirectory	Cria um novo diretório
rmdir	RemoveDirectory	Remove um diretório vazio
link	(nenhuma)	Win32 não dá suporte a ligações
unlink	DeleteFile	Destrói um arquivo existente
mount	(nenhuma)	Win32 não dá suporte a mount
umount	(nenhuma)	Win32 não dá suporte a mount
chdir	SetCurrentDirectory	Altera o diretório de trabalho atual
chmod	(nenhuma)	Win32 não dá suporte a segurança (embora o NT suporte)
kill	(nenhuma)	Win32 não dá suporte a sinais
time	GetLocalTime	Obtém o tempo atual

Leitura

- SISTEMAS OPERACIONAIS MODERNO 4ª edição
  - 1.5 Conceitos de sistemas operacionais
  - 1.6 Chamadas de sistema

Dúvidas?