QXD0013 - Sistemas Operacionais

Chamadas de Sistema, Estruturas de Sistemas Operacionais, Introdução a Processos

Thiago Werlley Bandeira da Silva[1]

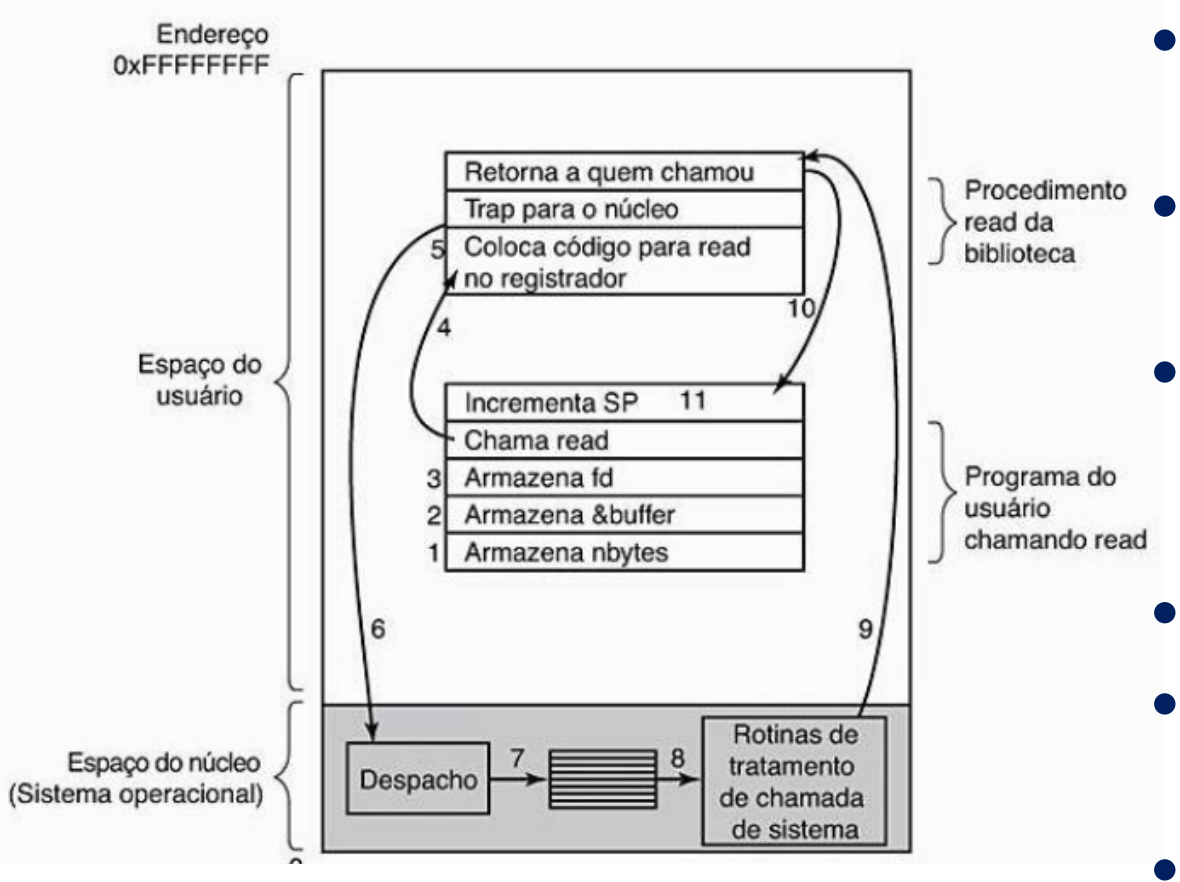
[1]Universidade Federal do Ceará, Brazil

28/03/2022

Chamadas de Sistema

- Os sistemas operacionais têm duas funções principais: fornecer **abstrações** para os programas de usuários e **gerenciar os recursos** do computador.
- As chamadas do sistema diferem de um SO para outro
- POSIX: interface mínima
- Muitas vezes expressas em código assembly
- Bibliotecas de rotinas
- Função TRAP (troca de modo)
- Exemplo: comando read
  - 3 parâmetros: arquivo, buffer, tamanho
  - Endereço do bloco de dados
  - contador = read(arq,&buffer,nbytes)
  - Em caso de falha: retorna -1 e código de erro atribuido a errno (variável global)
- Programas devem verificar resultado de uma chamada de sistema

Chamadas de Sistema



- Armazena parâmetros na pilha
- Chamada a rotina da biblioteca
- Coloca o número da chamada em um local esperado pelo SO
- Executa instrução TRAP
- Tabela $\rightarrow$ rotina de tratamento
- Retorno à rotina da biblioteca
- Programa limpa a pilha

Exemplo: Chamadas para Gerenciamento de Processos

- Chamada fork: criação de processo filho
  - Variáveis iguais após criação
  - Mudanças independentes
- Exemplo: shell
  - Lê comando do terminal
  - Cria processo filho
  - Expera que o filho execute o comando (waitpid)
  - Após terminado o processo filho, executa próximo comando

Exemplo: Chamadas para Gerenciamento de Processos

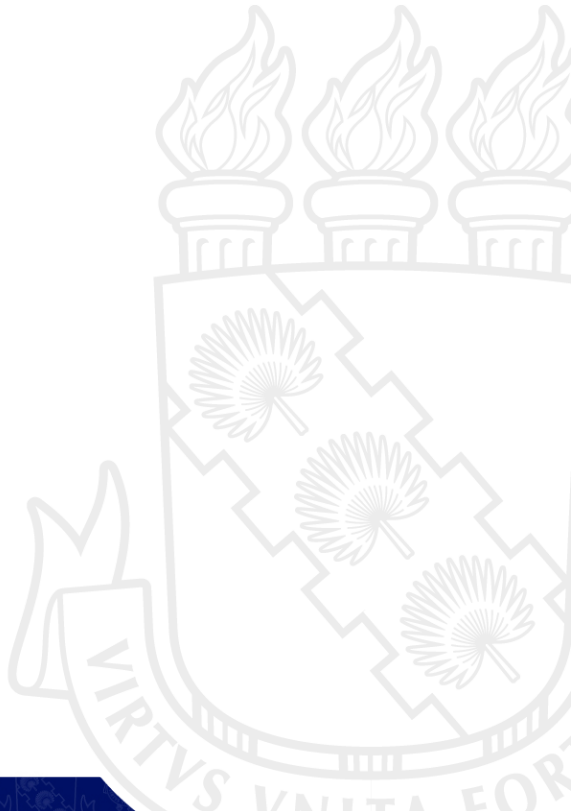
```
#define TRUE 1

while (TRUE) {                                  /* repeat forever */
        type_prompt( );                         /* display prompt on the screen */
        read_command(command, parameters);      /* read input from terminal */

        if (fork( ) != 0) {                     /* fork off child process */
             /* Parent code. */
             waitpid(–1, &status, 0);           /* wait for child to exit */
        } else {
             /* Child code. */
             execve(command, parameters, 0);    /* execute command */
        }
}
```

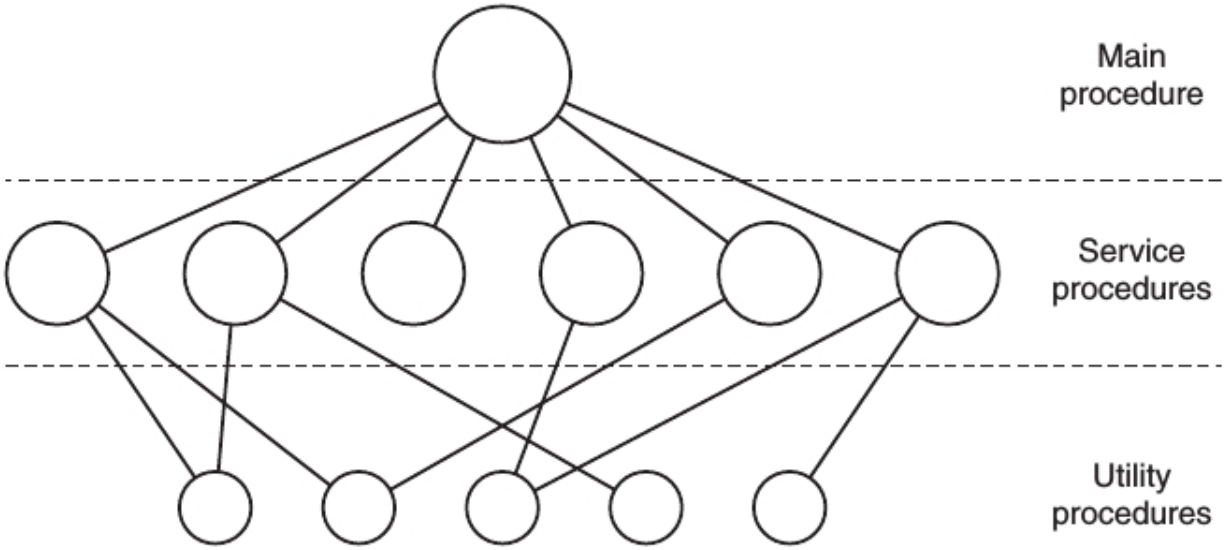
Chamadas: Windows x UNIX

- Chamadas de sistema $\neq$ Rotinas de bibliotecas
- UNIX $\rightarrow$ Quase relação 1 para 1
- Windows
  - Desacoplamento entre chamadas e rotinas
  - Rotinas encapsuladas na API Win32
  - Permite mudanças nas chamadas
  - Difícil distinção entre modos de execução



Estrutura de Sistemas Operacionais

- Sistemas Monolíticos
  - Único programa em modo núcleo
  - Possível chaveamento de modo
  - Rotinas podem chamar umas às outras
  - Dificuldade de compreensão
  - Estrutura básica (programa principal,rotinas de serviço e utilitárias)



Estrutura de Sistemas Operacionais

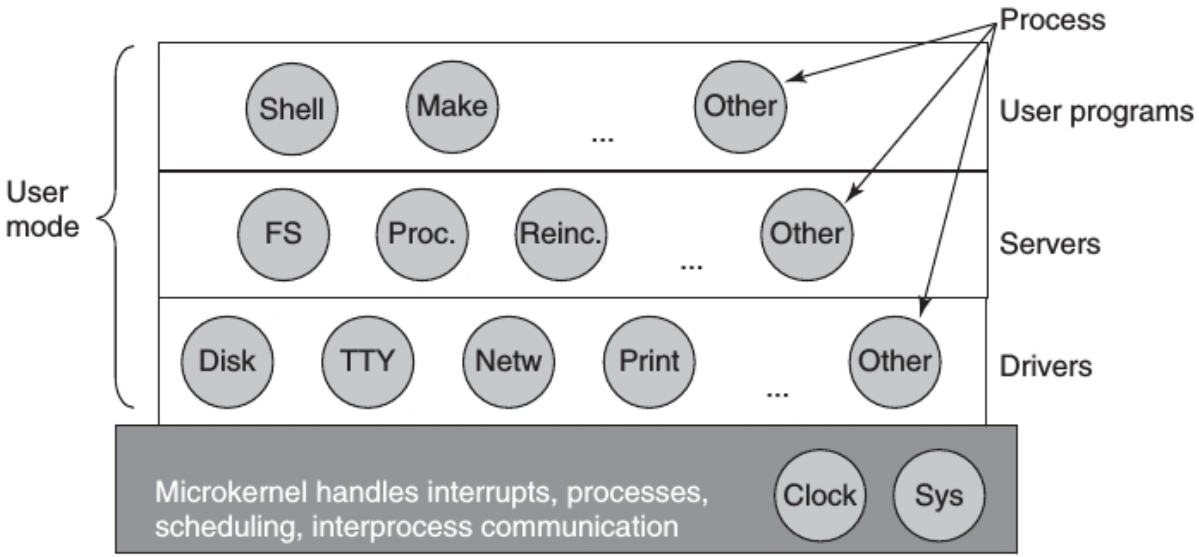
- Sistemas em Camadas
  - Hierarquia de camadas
    - Primeiro sistema: THE (6 camadas/ainda único programa)

Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

  - MULTICS: aneis cocêntricos, rotinas protegidas

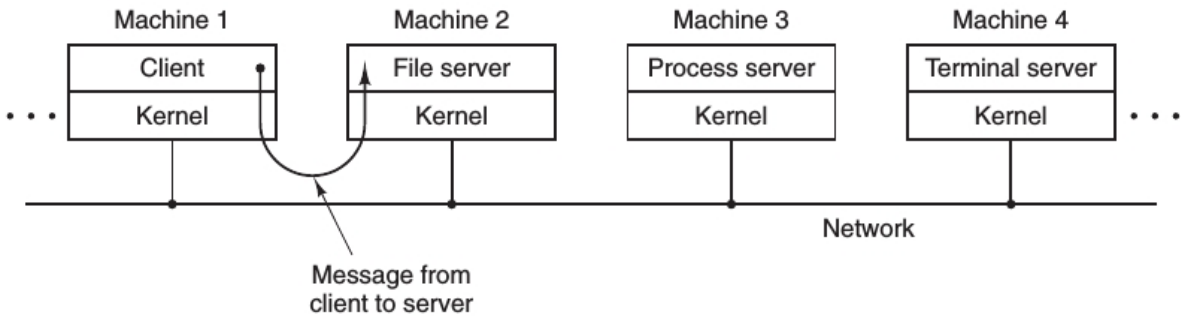
Estrutura de Sistemas Operacionais

- Sistemas em Micronúcleos
  - Subdividir em pequenos módulos
  - Apenas um em modo núcleo
  - Aumentar confiabilidade
  - Comuns em sistemas críticos
  - Mecanismo x Política
  - Exemplo: MINIX



Estrutura de Sistemas Operacionais

- Cliente-Servidor
  - Variação do micronúcleo
  - Duas classes de processos
  - Cliente: requisita
  - Servidor: fornece
  - Troca de mensagens
  - Abstração aplicável à redes

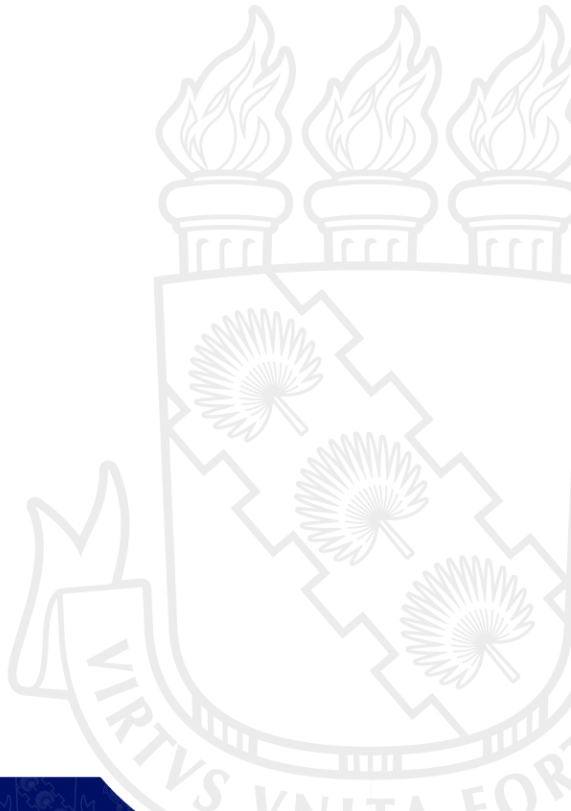


Estrutura de Sistemas Operacionais

- Máquinas Virtuais
  - Motivação: multiprogramação (OS/360)
  - Máquina Virtual: cópia exata do hardware
  - Monitor de Máquina Virtual (hypervisor)
  - Suporte a múltiplas máquinas virtuais
  - Flexibilidade x desempenho
  - Muito popular recentemente (hospedagem de sites)
  - Variação: Exonúcleo (não replica hardware)

Introdução a Processos

- Computadores modernos $\rightarrow$ várias ações simultâneas
- Exemplo: Servidor Web
  - Solicitações de páginas
  - Cache ou leitura do disco
  - Acesso ao disco muito lenta (visão da CPU)
  - Solicitações durante leitura do disco
- Necessidade de modelar e controlar
- Chaveamento da CPU: ilusão de paralelismo



Modelo de Processos

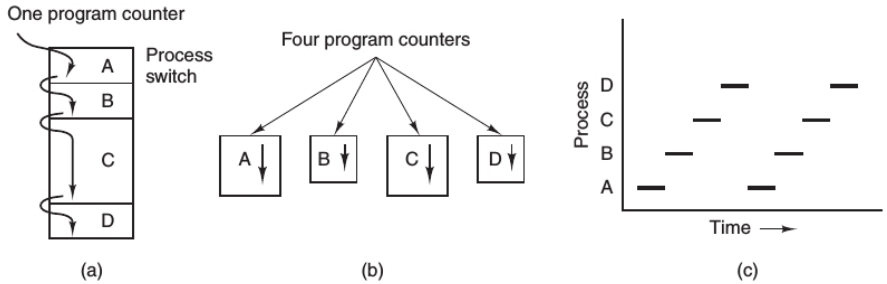
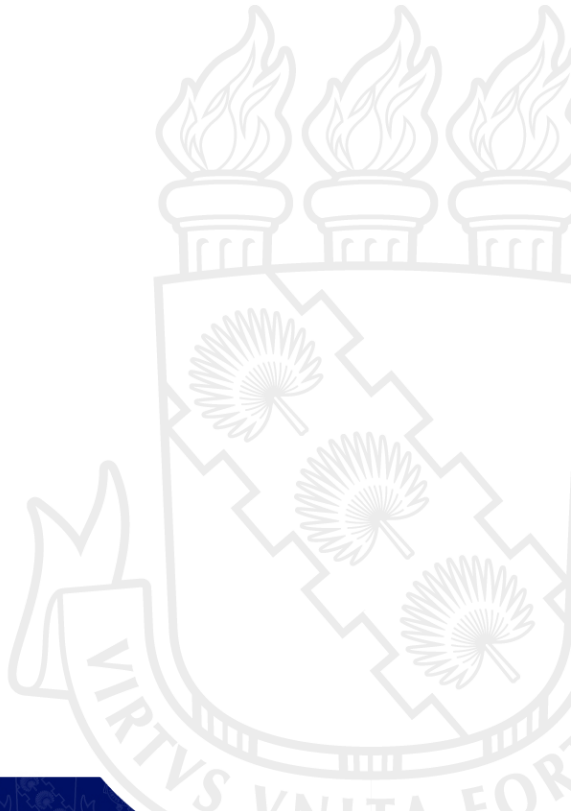


Figure 2-1. (a) Multiprogramming four programs. (b) Conceptual model of four independent, sequential processes. (c) Only one program is active at once.

- Processo → atividade:
  - Programa (lista de instruções)
  - Contador de programa (endereço de instrução)
  - Registradores
  - Variáveis
- Apenas um contador de programa real (físico)
- Contador lógico carregado no físico
- Escalonamento: regras de controle (computação não uniforme)
- Requisitos de tempo-real → capacidades especiais do SO
- Dois programas iguais → processos distintos

Criação de Processos

- Sistemas simples $\rightarrow$ ao ser ligado
- Sistemas de propósito geral: 4 eventos
- Início do sistema
  - Foreground x Background (daemons)
- Por outro processo
  - Divisão de tarefas
- Pelo usuário
  - Comando ou clique no mouse
- Tarefa em lote (batch)
  - Submissão de tarefas



Criação de Processos

- Em todos os casos ocorre uma chamada de sistema
  - Pai inicia processo Filho (fork)
  - Processo Filho executa programa (execve)
- Porque dois passos?
  - Permite filho executar ações (entre fork e execve)
  - Redirecionar entradas e saídas (padrão e de erros)
- Windows: apenas um passo (CreateProcess)
- Espaços de endereçamento distintos
  - Filho uma cópia do pai inicialmente
  - Não compartilhada

Término de Processos

- 4 eventos
- Saída normal (voluntário)
  - Trabalho concluido
  - Chamada informa ao SO (exit no UNIX e ExitProcess no Windows)
  - Exemplo: compilação terminada ou comando do usuário
- Saída por erro (voluntário)
  - Programa encontra um erro
  - Raro quando baseados em janela
  - Exemplo: compilar arquivo inexistente

Término de Processos

- Erro fatal (involuntário)
  - Erros no programa
  - Instrução ilegal
  - Tratamento de exceção: sinalização ao SO
  - Exemplo: referência a memória inexistente, divisão por zero
- Por outro processo (involuntário)
  - Chamada para finalizar processo
  - Exemplo: kill (UNIX) e TerminateProcess (Windows)
- Alguns casos: filhos finalizados junto com o pai

Questões:

1. **Para cada uma das chamadas de sistema a seguir, dê uma condição que a faça falhar: fork, exec e unlink.**
   - ○

Questões:

1. **Para cada uma das chamadas de sistema a seguir, dê uma condição que a faça falhar: fork e exec.**
   - A chamada de sistema "fork" falhará caso não exista espaços livres na tabela de processos. O "exec" falhará caso não exista o nome do arquivo passado como parâmetro.

Questões:

2. **A chamada count = write(fd, buffer, nbytes); pode retornar qualquer valor em count fora nbytes? Se a resposta for sim, por quê?**
   -

Questões:

2 **A chamada count = write(fd, buffer, nbytes); pode retornar qualquer valor em count fora nbytes? Se a resposta for sim, por quê?**

- Sim, pode haver retorno de valor diferente de nbytes. Isso se justifica pelo fato de que a chamada pode falhar caso o arquivo seja inválido ou não possa ser lido, retornando o valor -1. Outra forma para que o nbytes não seja retornado, se deve ao fato de que caso o disco esteja cheio, não será possível escrever o número de bytes que foram solicitados, ou seja, o final do arquivo pode ter sido encontrado antes.

Questões:

3. **A rotina de biblioteca é chamada read e a chamada de sistema em si é chamada read. É fundamental que ambas tenham o mesmo nome? Se não, qual é a mais importante?**
   - ○

Questões:

3 **A rotina de biblioteca é chamada read e a chamada de sistema em si é chamada read. É fundamental que ambas tenham o mesmo nome? Se não, qual é a mais importante?**

- Não é necessário que a rotina de biblioteca e a chamada de sistema tenham o mesmo nome. O nome da rotina de biblioteca é o mais importante, já que este consta de forma "nativa" ao sistema operacional. O que a rotina de biblioteca faz é ler traps para o kernel colocando o número da chamada de sistema em um registro ou em uma pilha. Esse número é indexado em uma tabela de ponteiros.

Questões:

4 **Para um programador, uma chamada de sistema parece com qualquer outra chamada para uma rotina de biblioteca. É importante que um programador saiba quais rotinas de biblioteca resultam em chamadas de sistema? Em quais circunstâncias e por quê?**

Questões:

4 **Para um programador, uma chamada de sistema parece com qualquer outra chamada para uma rotina de biblioteca. É importante que um programador saiba quais rotinas de biblioteca resultam em chamadas de sistema? Em quais circunstâncias e por quê?**

- Sim. Quando as rotinas de biblioteca resultam em chamadas de sistema, o programa será executado de forma mais lenta do que as rotinas de biblioteca que não resultam em chamadas de sistema. Isso acontece porque a cada chamada de sistema há uma mudança de contexto do usuário para o kernel, e isso toma tempo o que acarreta em uma queda de desempenho.

Questões:

5 **Qual é a finalidade das chamadas de sistema?**

- ○

Questões:

5 **Qual é a finalidade das chamadas de sistema?**

- As chamadas de sistema têm a finalidade de fornecer uma interface entre os processos e o sistema operacional. São a partir delas que se tem acesso as rotinas do sistema. As chamadas de sistemas são usadas pelos programas para a solicitação da execução de algum serviço junto ao núcleo (kernel) do sistema operacional. Além disso, a criação e finalização dos processos dependem da permissão dessas chamadas de sistema. É através dessas chamadas de sistema que são definidos os acessos aos recursos de baixo nível como alocação de memória, periféricos e arquivos.

Questões:

6 **Que chamadas de sistema têm de ser executadas por um interpretador de comandos ou shell para iniciar um novo processo?**

-

Questões:

6 **Que chamadas de sistema têm de ser executadas por um interpretador de comandos ou shell para iniciar um novo processo?**

- Com base no sistema UNIX, o interpretador de comandos deve executar uma chamada de sistema fork() que cria o processo filho idêntico ao pai exceto por alguns atributos e recursos), onde em seguida o programa selecionado é carregado na memória e executado através da chamada de sistema exec(). Com base no Windows, o shell deve executar a chamada de sistemas CreateProcess().

Questões:

7 **Quais são as vantagens e desvantagens do uso da mesma interface de chamadas de sistema para manipular tanto arquivos quanto dispositivos?**

- ○

Questões:

7 **Quais são as vantagens e desvantagens do uso da mesma interface de chamadas de sistema para manipular tanto arquivos quanto dispositivos?**

- Vantagem: fácil de adicionar um novo dispositivo de driver implementando o código específico do hardware para suportar esta interface, beneficiando o desenvolvimento do código do programa usuário, onde este já pode ser escrito para acessar dispositivos e arquivos da mesma maneira.
  Desvantagem: pode ser difícil capturar a funcionalidade de certos dispositivos no contexto.