

# **Sistemas Operacionais**

**SEMANA 4**

# Sistemas Operacionais

## Tópicos

- Tarefas
  - Contexto
  - Troca de Contexto
  - Processos
  - Threads

# Sistemas Operacionais

## Tarefas

- Contexto
  - Toda tarefa possui um estado bem definido
    - Esse estado se refere aos seus valores de variáveis, posição do contador de programa e recursos que utiliza
  - Um determinado instante de execução da tarefa também é conhecido como **contexto**
  - O contexto também envolve o estado interno do processador
    - Contador de programa, apontador de pilha e demais registradores

# Sistemas Operacionais

## Tarefas

- Contexto
  - Como funciona no núcleo do SO?
    - O núcleo possui um *descriptor* associado a cada tarefa presente no sistema
    - Este *descriptor* é uma estrutura de dados (**TCB – Task Control Block** ou **PCB – Process Control Block**) que representa essa tarefa

# Sistemas Operacionais

## Tarefas

- Contexto
  - Dados do **TCB**
    - Identificador da tarefa
    - Estado da tarefa (nova, pronta...)
    - Informações de contexto do processador
      - Valores dos registradores
    - Lista de áreas da memória usadas pela tarefa
    - Lista de arquivos abertos, conexões de rede e recursos
    - Informações de gerência e contabilização
      - Prioridade, usuário proprietário, data de início, tempo de processamento já decorrido, volume de dados, etc...

# Sistemas Operacionais

## Tarefas

- Contexto
  - Os *descritores* são organizados em lista ou vetores
    - Lista de tarefas prontas
    - Lista de tarefas aguardando acesso ao HD
    - Lista....

# Sistemas Operacionais

## Tarefas

- Troca de Contexto
  - Na aula passada vimos que os sistemas que utilizam *Time sharing* (compartilhamento de tempo) alternam as tarefas em execução após um dado tempo (**quantum**)
  - Para suspender uma tarefa e retomar a execução de outra é necessário salvar o contexto da tarefa em seu TCB

# Sistemas Operacionais

## Tarefas

- Troca de Contexto
  - O ato de suspender uma tarefa e reativar outra é chamado de **Troca de Contexto**
  - As **Trocas de Contexto** envolvem o conhecimento do processador
    - Cada processador pode ter flags e registradores específicos
    - Exemplo no Linux (Intel x64):
      - `arch/x86/kernel/process_64.c`
      - `arch/x86/entry/entry_64.S`



# Sistemas Operacionais

## Tarefas

- Troca de Contexto
  - Para a troca de contexto, utilizamos dois componentes do SO
    - Despachante
    - Escalonador

# Sistemas Operacionais

## Tarefas

- Troca de Contexto
  - **Despachante (Executivo)**
    - Conjunto de Rotinas mecânicas
    - Envolvem armazenar e recuperar informações do TCB
  - **Escalonador (Scheduler)**
    - Escolha da próxima tarefa a entrar em execução
    - Leva em conta prioridades, tempo de vida, tempo de processamento restante

# Sistemas Operacionais

## Tarefas

- Troca de Contexto

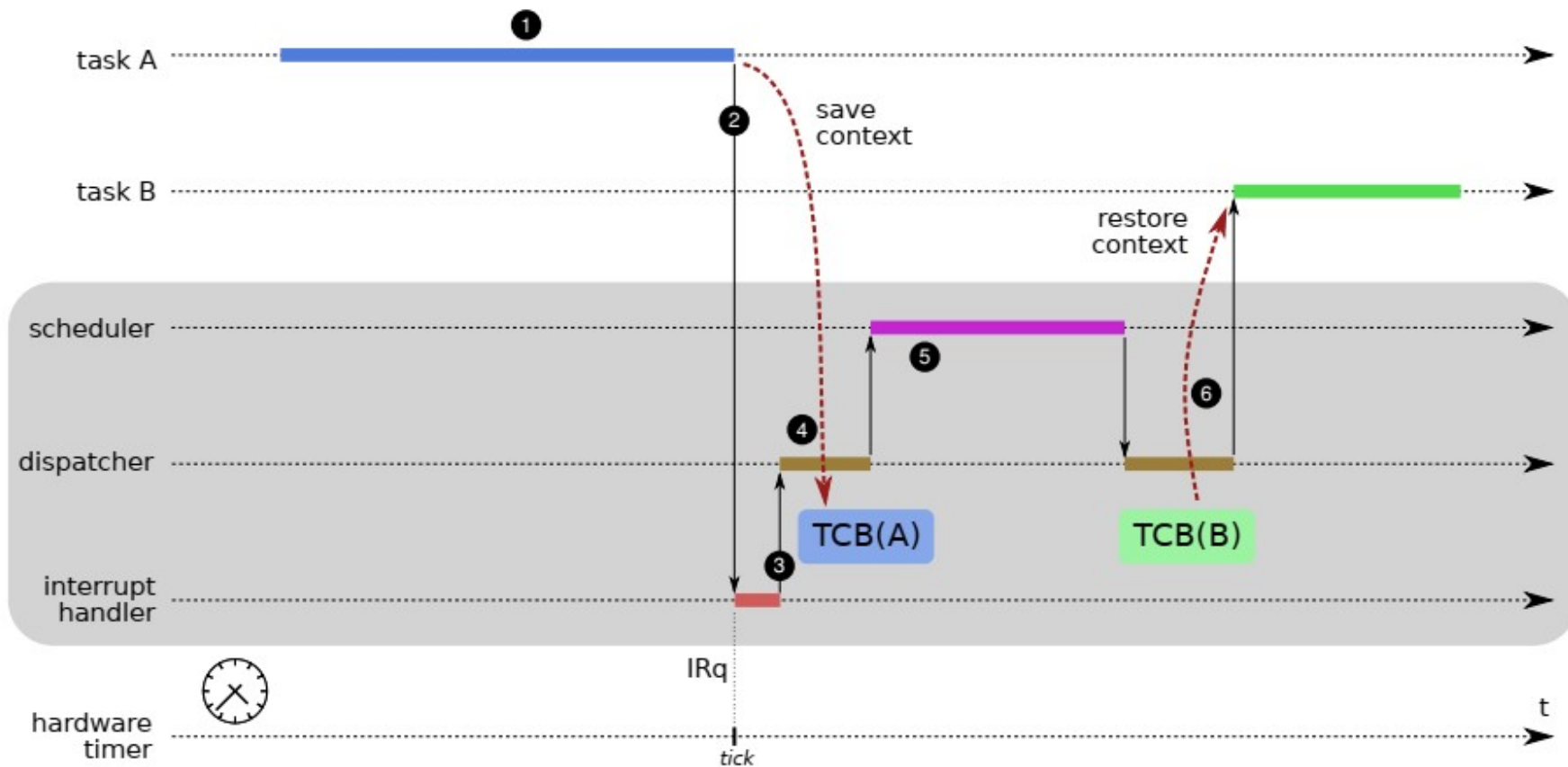


Figura 5.1: Passos de uma troca de contexto.

## Tarefas

- Troca de Contexto

1. Tarefa *A* está executando
2. *Interrupção do temporizador* e a execução desvia para a *rotina de tratamento*, no núcleo
3. *Rotina de tratamento* ativa o **despachante**
4. **Despachante** salva o estado da tarefa *A* em seu TCB e atualiza suas informações de gerência
5. **Despachante** consulta o escalonador para escolher a próxima tarefa a ativar (*B*)
6. **Despachante** resgata o estado da tarefa *B* de seu TCB e a reativa

## Tarefas

- Troca de Contexto
  - Motivos de troca de Contexto
    - Final do quantum
    - Evento de um periférico
      - Interrupção pelo controlador
    - Chamada de sistema
      - Emitida pela própria tarefa em execução
    - Erro de execução
      - Exceção

# Sistemas Operacionais

## Tarefas

- Troca de Contexto
  - Quanto maior o número de trocas de contexto pior será a eficiência de nosso SO
  - Quanto menor o número de trocas de contexto e menor o tempo de duração de cada troca, mais tempo sobra para execução de tarefas

# Sistemas Operacionais

## Tarefas

- Troca de Contexto
  - Medida de eficiência

$$\varepsilon = \frac{t_q}{t_q + t_{tc}}$$

- $t_q$  – média do *quantum*
- $t_{tc}$  – média de trocas de contexto

## Tarefas

- Troca de Contexto
  - Medida de eficiência
    - Exemplo
      - $t_q = 10\text{ms}$  e  $1\text{ms}$
      - $t_{tc} = 100\mu\text{s}$  e  $100\mu\text{s}$
      - $= 99\%$  e  $91\%$



# Sistemas Operacionais

## Tarefas

- Troca de Contexto
  - Medida de eficiência
    - Existem fatores que impactam nessa eficiência
      - Muitas tarefas ao mesmo tempo (maior  $t_{tc}$ )
      - Aplicações com muito I/O
        - » Saem do processador antes do final do *quantum*

# Sistemas Operacionais

## Tarefas

- Troca de Contexto
  - Muitos SOs não executam o escalonador a cada troca de contexto
    - Escalonador é uma tarefa que possui alto custo computacional
  - O despachante ativa a primeira tarefa pronta da fila
  - Escalonador será executado em uma troca de contexto quando há necessidade de reordenar a fila de tarefas prontas

# Sistemas Operacionais

## Tarefas

- Atividade
  - Troca de contexto é uma tarefa efetuada pelo Sistema Operacional na gerência de tarefas. A troca de contexto consiste em:
    - (A) trocar o usuário logado no Sistema Operacional, para que outro usuário possa utilizá-lo sem interferência nas informações do usuário anterior.
    - (B) interromper a execução de aplicativos críticos.
    - (C) salvar informações de uma tarefa para que o processador possa ser entregue a outra, carregando seu contexto.
    - (D) recarregar o contexto do usuário para restaurar o estado da máquina.
    - (E) trocar a tarefa que gerencia as impressoras instaladas na máquina.

# Sistemas Operacionais

## Tarefas

- Atividade
  - O que significa troca de contexto? E o que exatamente compreende o contexto de um processo? Que componente do sistema operacional realiza a troca de contexto?
  - Explique o que é, para que serve e o que contém um TCB - Task Control Block.
  - O que é o contexto e para que ele serve?

# Sistemas Operacionais

## Tarefas

- Processos
  - Existem diversas formas do SO implementar uma tarefa
    - Uma delas é através de **PROCESSOS**
  - **PROCESSOS**
    - *Unidade de contexto*
    - Um contêiner de recursos utilizados por uma ou mais tarefas para sua execução
      - Áreas de memória: código, dados, pilha
      - Informações de contexto: TCB
      - Descritos de recursos do núcleo: arquivos, conexões de rede, etc

# Sistemas Operacionais

## Tarefas

- Processos
  - Podem conter uma ou mais tarefas compartilhando seus recursos
  - Processos são isolados entre si
    - Isolamento de áreas de memória
    - Níveis de operação
    - Chamada de sistema
  - Tarefas de um determinado processo não acessam recursos atribuídos a outro processo

# Sistemas Operacionais

## Tarefas

- Processos

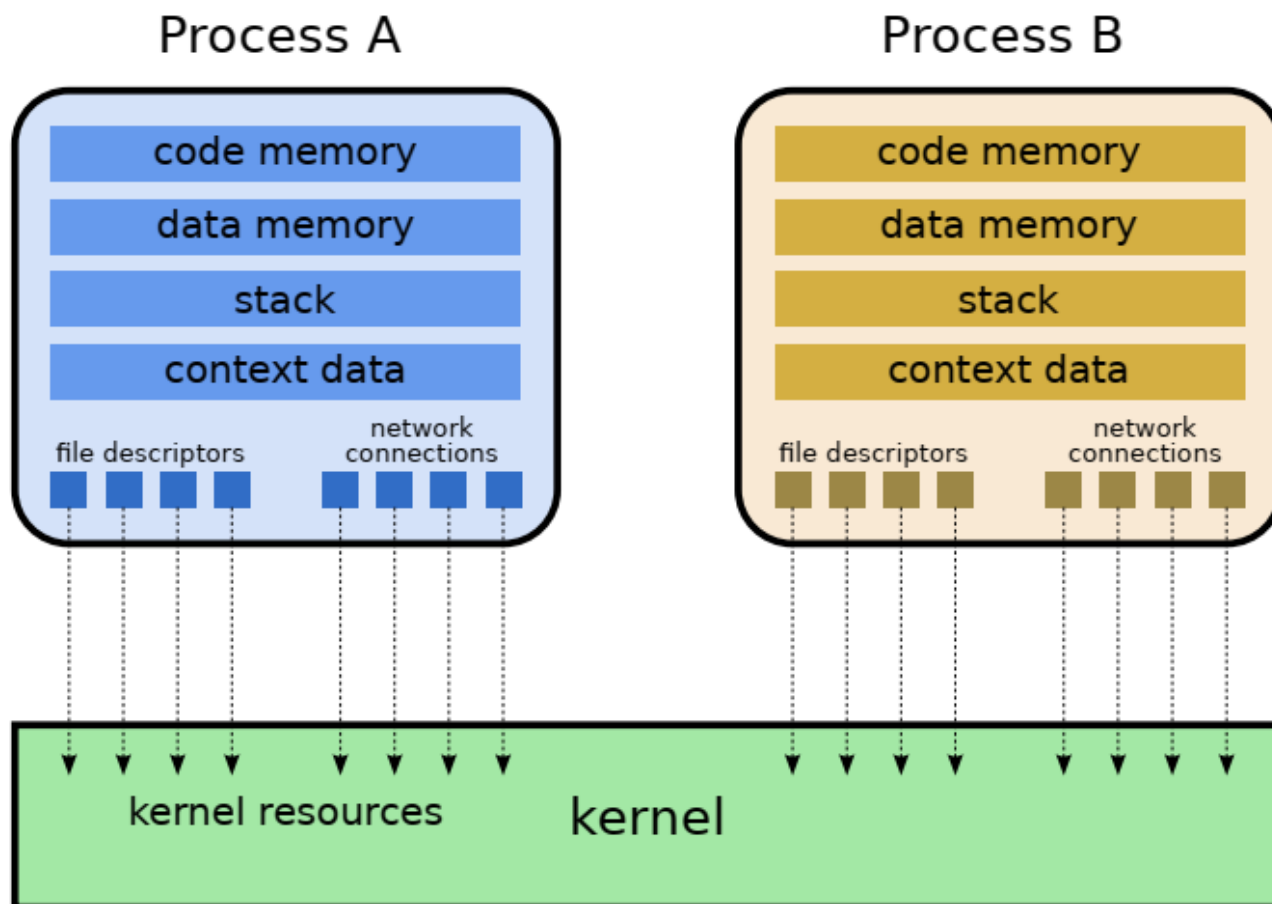


Figura 5.2: O processo visto como um contêiner de recursos.

# Sistemas Operacionais

## Tarefas

- Processos
  - Inicialmente cada processo representa uma única tarefa
  - Corresponde a execução de um programa sequencial
    - Exemplo: Iniciou a execução das instruções da função `main()` de um programa em C
  - Para que novas tarefas sejam criadas dentro deste processo o **desenvolvedor** deve solicitar ao núcleo a criação de tarefas adicionais: **THREADS**



# Sistemas Operacionais

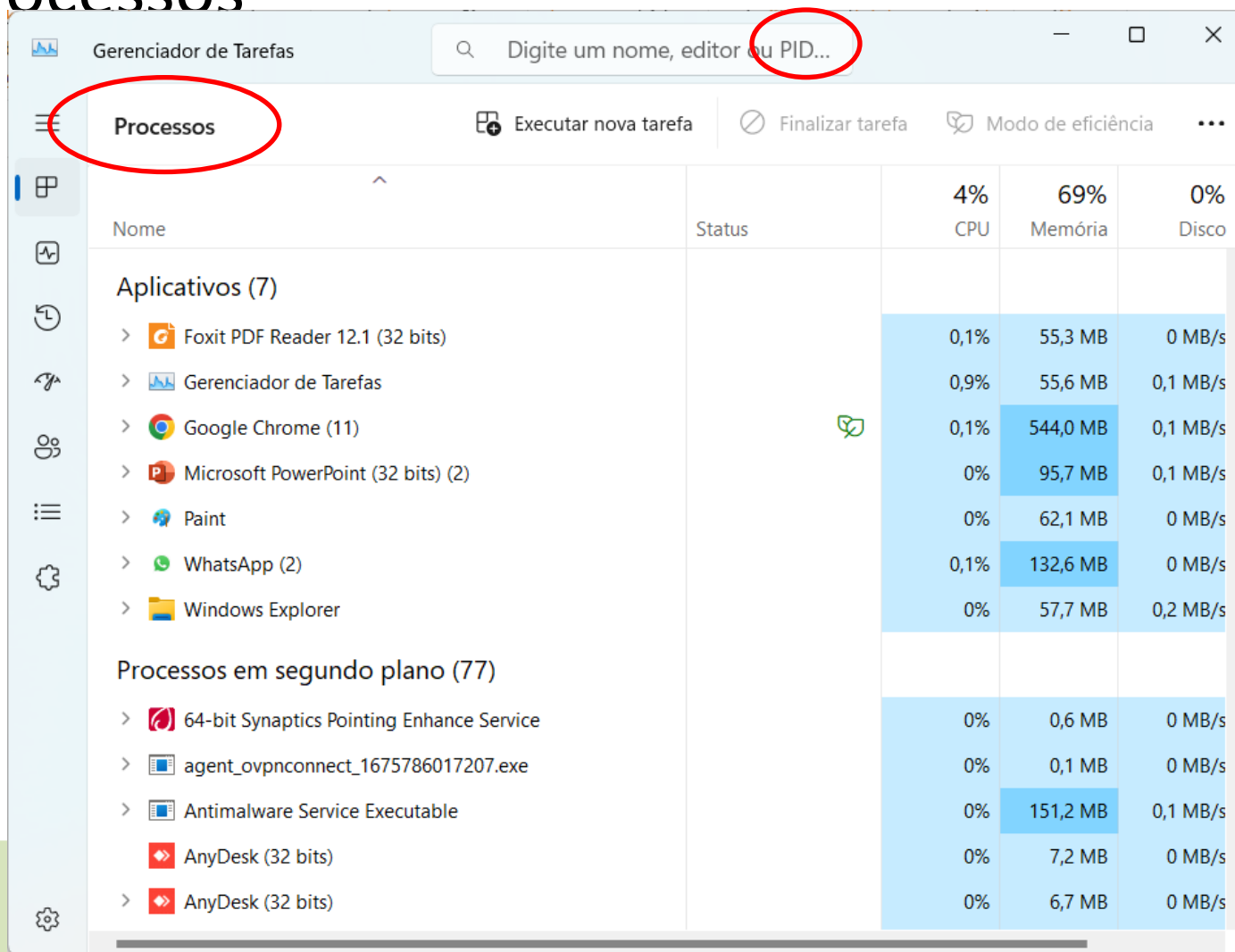
## Tarefas

- Processos
  - **O PCB** (já citado anteriormente) é o descritor do processo
    - Armazena informação do processo
    - Identificador do processo (PID – Process Identifier)
    - Usuário (proprietário do processo)
    - Prioridade
    - Data início
    - Caminho do arquivo executado
    - Áreas de memória
    - Arquivos abertos...

# Sistemas Operacionais

## Tarefas

- Processos



Gerenciador de Tarefas					
Digite um nome, editor ou PID...					
Processos					
Executar nova tarefa Finalizar tarefa Modo de eficiência					
Nome	Status	4% CPU	69% Memória	0% Disco	
Aplicativos (7)					
> Foxit PDF Reader 12.1 (32 bits)		0,1%	55,3 MB	0 MB/s	
> Gerenciador de Tarefas		0,9%	55,6 MB	0,1 MB/s	
> Google Chrome (11)		0,1%	544,0 MB	0,1 MB/s	
> Microsoft PowerPoint (32 bits) (2)		0%	95,7 MB	0,1 MB/s	
> Paint		0%	62,1 MB	0 MB/s	
> WhatsApp (2)		0,1%	132,6 MB	0 MB/s	
> Windows Explorer		0%	57,7 MB	0,2 MB/s	
Processos em segundo plano (77)					
> 64-bit Synaptics Pointing Enhance Service		0%	0,6 MB	0 MB/s	
> agent_ovpnconnect_1675786017207.exe		0%	0,1 MB	0 MB/s	
> Antimalware Service Executable		0%	151,2 MB	0,1 MB/s	
> AnyDesk (32 bits)		0%	7,2 MB	0 MB/s	
> AnyDesk (32 bits)		0%	6,7 MB	0 MB/s	

# Sistemas Operacionais

## Tarefas

- Processos
  - Desta forma, associa-se o TCB da tarefa ao seu processo
    - Simplifica o TCB
    - Armazena apenas o identificador da tarefa
    - Os registradores do processador
    - Referência para o processo
  - A troca de contexto de duas tarefas dentro do mesmo processo é mais simples e rápida

# Sistemas Operacionais

## Tarefas

- Gestão de Processos
  - O SO gerencia a lista de processos
    - Processos são criados e destruído várias vezes durante a operação do SO
    - Essas operações (criação e destruição) são fornecidas pelo SO
      - Chamadas de sistema

# Sistemas Operacionais

## Tarefas

- Gestão de Processos
  - Exemplo em sistema UNIX
    - Criação de novos processos
      - Chamada a `fork()`: função que cria uma réplica de um processo
      - Espaços de memória do processo inicial são copiados para o novo processo
      - Inclui código de tarefas associadas, descritos de arquivos e demais recursos
      - A chamada de sistema `fork()` é invocada por um processo,
        - » Dois processos recebem retorno quem chamou e o novo processo

# Sistemas Operacionais

## Tarefas

- Gestão de Processos
  - Exemplo em sistema UNIX

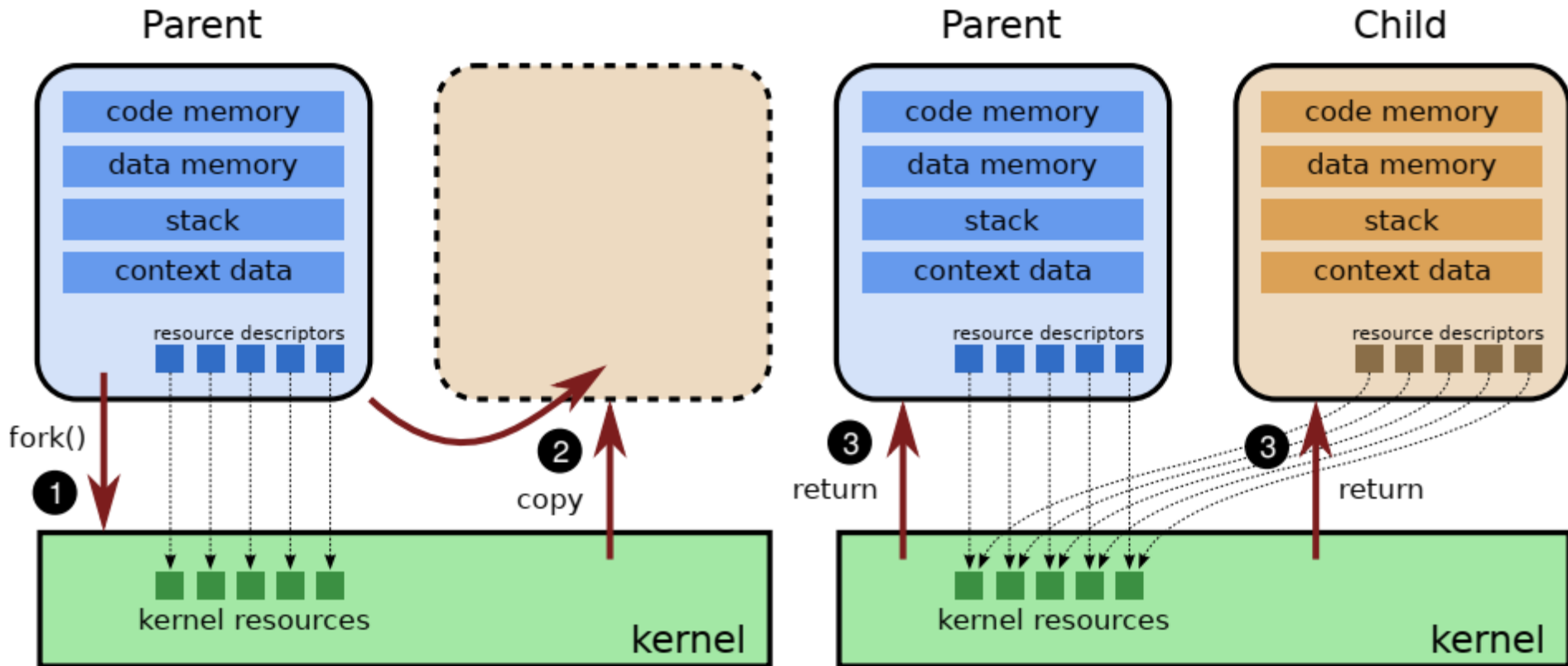


Figura 5.3: Execução da chamada de sistema `fork()`.

# Sistemas Operacionais

## Tarefas

- Hierarquia de processos
  - Quando um novo processo é criado ele é *filho* de outro processo (*pai*)
    - A cada novo processo forma-se uma *árvore de processos*
    - Usada para gerencia os processos ligados a uma mesma aplicação
    - Caso um processo se encerre, todos os sub-processos serão encerrados

# Sistemas Operacionais

## Tarefas

- **THREAD**

- Inicialmente os SOs suportavam apenas uma tarefa por processo
  - Com a complexidade das novas aplicações esta limitação se torna inconveniente
  - Imagine você ter que aguardar enquanto sua página de internet carrega e o navegador fica em um estado “bloqueado”
    - O ideal é que você consiga operar outras funções do navegador sem atrapalhar o carregamento da página



## Tarefas

- **THREAD**

- Para estas novas aplicações foi criado o conceito de **Thread**
- Uma **Thread** é um fluxo de execução independente
  - Um processo possui uma ou várias threads
  - Cada thread executa seu próprio código
  - Compartilham recursos entre as threads de um mesmo processo

## Tarefas

- **THREAD**

- Uma Thread é caracterizada por possuir seu próprio código de execução
  - Mas também possui um pequeno contexto local
    - *Thread Local Storage (TLS)*
      - » Registradores do processador
      - » Pilha em memória
        - Variáveis locais
        - Chamadas de funções

## Tarefas

- **THREAD**

- As threads também são utilizadas para implementar fluxos de execução dentro do núcleo do SO
  - *Threads de Núcleo*
  - Incluem atividades internas do núcleo
    - Rotinas de drivers
    - Tarefas de gerência

# Sistemas Operacionais

## Tarefas

- **THREAD**

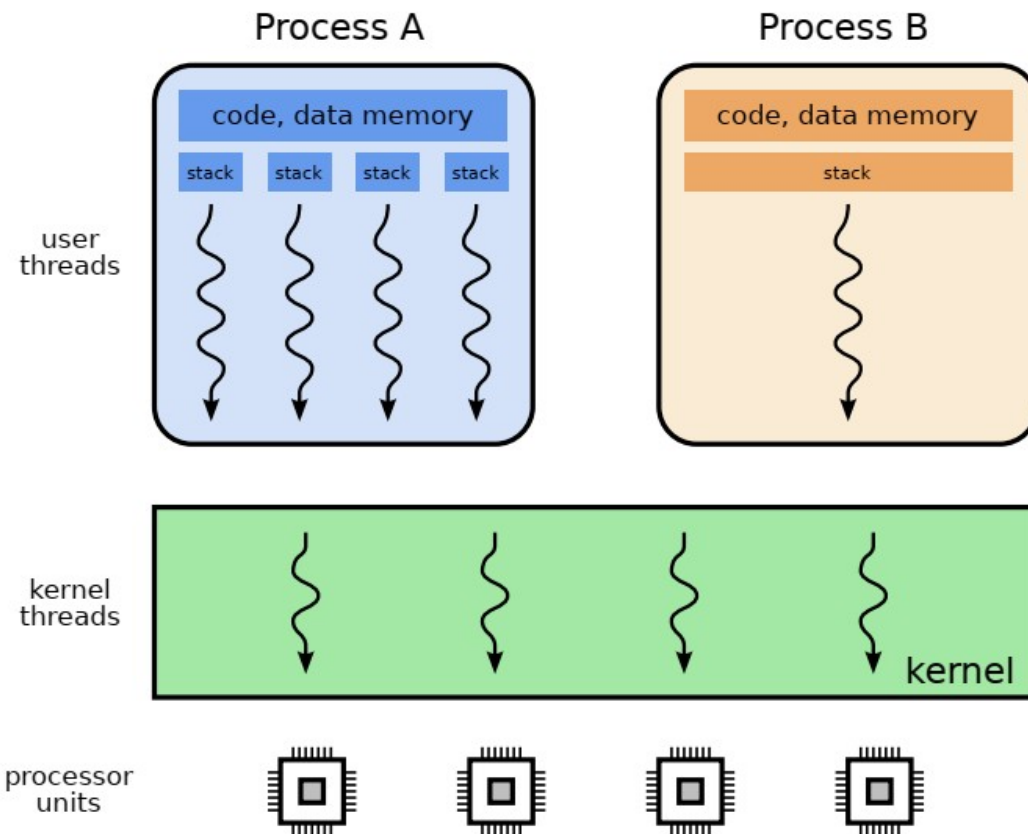


Figura 5.4: *Threads* de usuário e de núcleo.

# Sistemas Operacionais

## Tarefas

- **Modelos de Gerência de Threads**
  - Todas as threads são gerenciadas pelo núcleo do SO
    - São diversas as formas de gerência

## Tarefas

- **Modelos de Gerência de Threads**
  - Modelo N:1
    - SOs antigos permitiam apenas um fluxo de execução
    - Desenvolvedores criavam rotinas para salvar o contexto dentro do processo e controlavam as diferentes tarefas que executavam
    - O núcleo percebe apenas uma tarefa, mas a aplicação gerencia várias
      - MODELO N:1

# Sistemas Operacionais

## Tarefas

- **Modelos de Gerência de Threads**
  - Modelo N:1

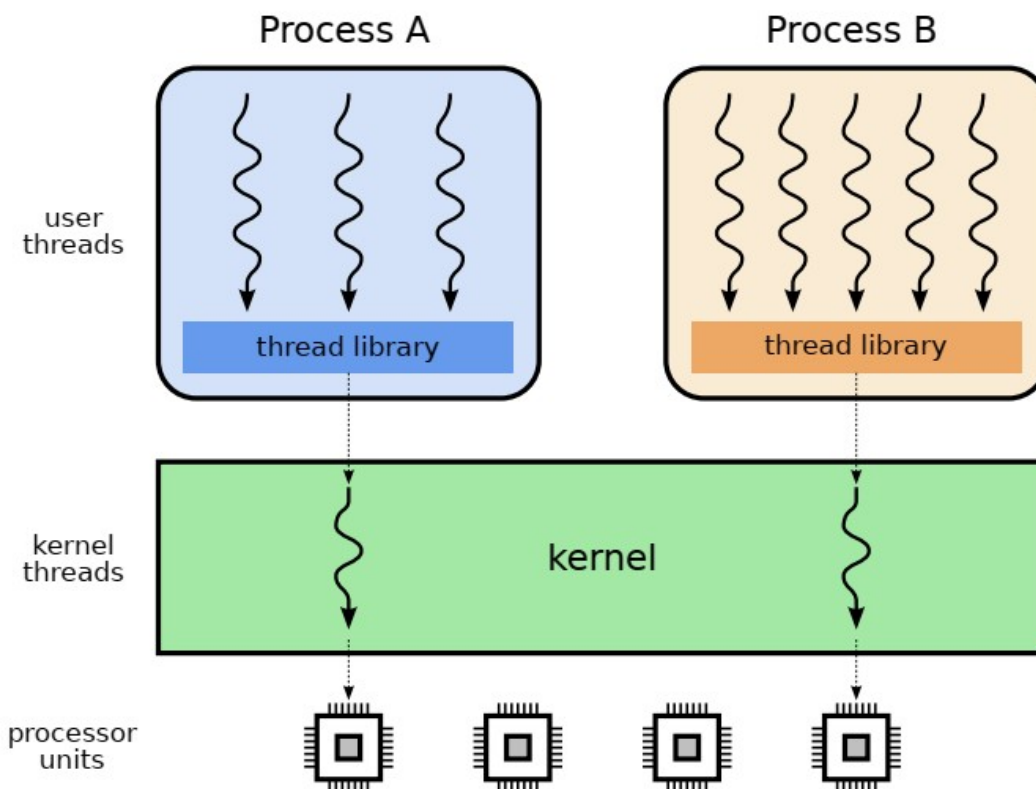


Figura 5.5: O modelo de *threads* N:1.

# Sistemas Operacionais

## Tarefas

- **Modelos de Gerência de Threads**
  - Modelo N:1
    - Problemas
      - Operações de entrada/saída são realizadas pelo núcleo
        - » Se uma thread solicitar uma operação desta, o processo inteiro ficará bloqueado (inclusive as demais threads)
      - As threads de núcleo dividem o tempo de processamento do sistema
        - » 100 threads de um processo recebem o mesmo tempo que 1 thread de outro processo
      - Threads de um mesmo processo não executam em paralelo mesmo se o computador possui mais processadores



## Tarefas

- **Modelos de Gerência de Threads**
  - Modelo 1:1
    - Aplicações *multithread*
    - Sistemas modernos gerencia uma thread de núcleo para cada thread de usuário

# Sistemas Operacionais

## Tarefas

- **Modelos de Gerência de Threads**

- Modelo 1:1

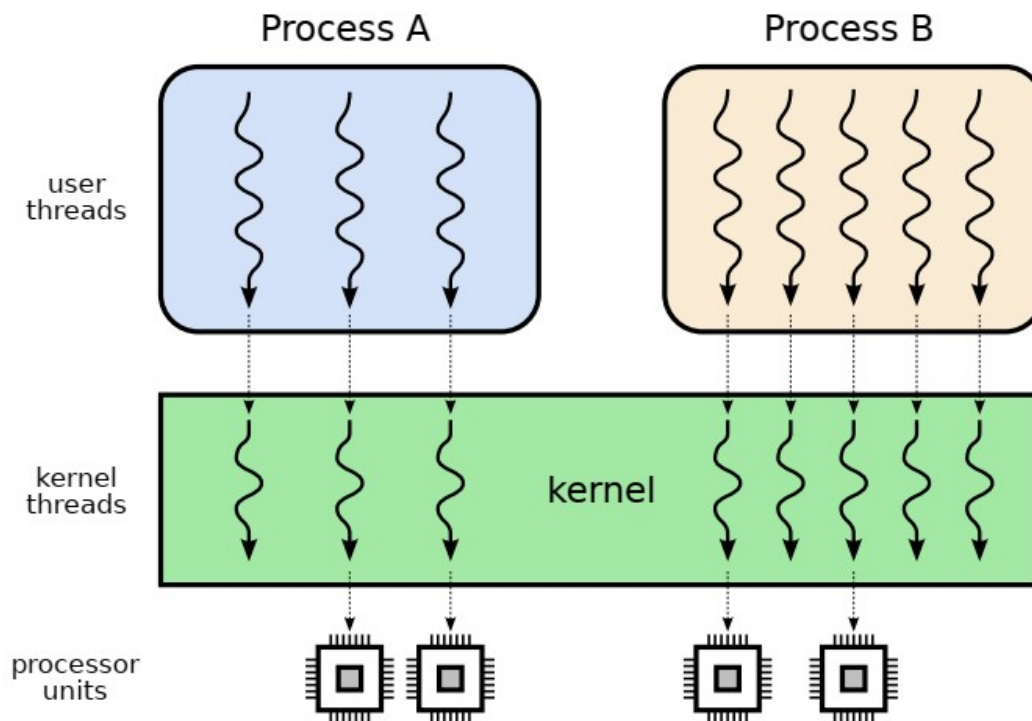


Figura 5.6: O modelo de *threads* 1:1.

## Tarefas

- **Modelos de Gerência de Threads**
  - Modelo 1:1
    - Problemas da N:1 são resolvidos
      - Somente a thread solicitante de operações de I/O que é bloqueada
      - A distribuição é por Thread e não por processo
    - Novos problemas
      - Criação de muitas threads impõe elevada carga ao núcleo
      - Inviabiliza aplicações com muitas tarefas

## Tarefas

- **Modelos de Gerência de Threads**
  - Modelo N:M
    - Para resolver o problema da abordagem 1:1
    - Modelo híbrido
    - Biblioteca gerencia um conjunto de N threads de usuário
      - Dentro do processo
    - M threads são criadas no núcleo
      - Sendo  $M < N$

# Sistemas Operacionais

## Tarefas

- **Modelos de Gerência de Threads**
  - Modelo N:M

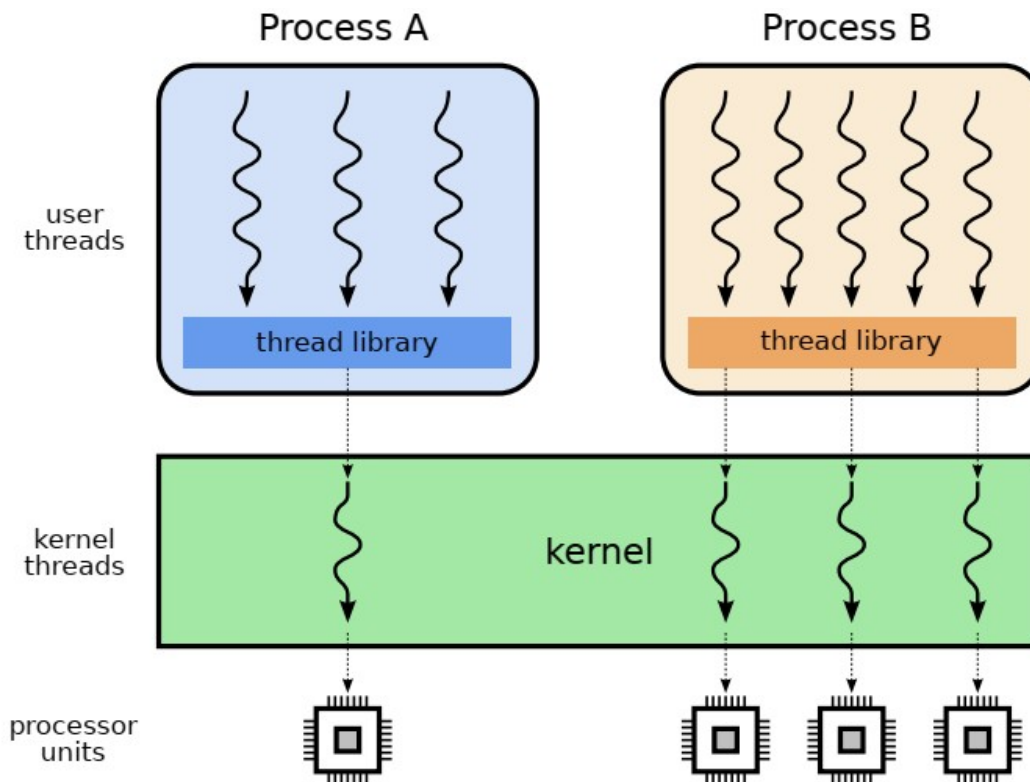


Figura 5.7: O modelo de *threads* N:M.

## Tarefas

- **Modelos de Gerência de Threads**
  - Modelo N:M
    - Alia as vantagens do 1:1
    - Escalabilidade do N:1
    - Problemas:
      - Maior complexidade para implementação
      - Maior custo para gerência dos threads de núcleo

# Sistemas Operacionais

## Tarefas

- **Comparação dos Modelos**

Modelo	N:1	1:1	N:M
Resumo	N <i>threads</i> do processo mapeados em uma <i>thread</i> de núcleo	Cada <i>thread</i> do processo mapeado em uma <i>thread</i> de núcleo	N <i>threads</i> do processo mapeados em $M < N$ <i>threads</i> de núcleo
Implementação	no processo (biblioteca)	no núcleo	em ambos
Complexidade	baixa	média	alta
Custo de gerência	baixo	médio	alto
Escalabilidade	alta	baixa	alta
Paralelismo entre <i>threads</i> do mesmo processo	não	sim	sim
Troca de contexto entre <i>threads</i> do mesmo processo	rápida	lenta	rápida
Divisão de recursos entre tarefas	injusta	justa	variável, pois o mapeamento <i>thread</i> → processador é dinâmico
Exemplos	GNU Portable Threads, Microsoft UMS	Windows, Linux	Solaris, FreeBSD KSE

# Sistemas Operacionais

## Tarefas

- **Processos vs Threads**
  - Um processo por tarefa
    - Caso o processo venha a falhar, apenas este processo será bloqueado
    - Cada processo pode ter seu próprio nível de acesso e permissões
      - Maior segurança pro sistema
    - Compartilhamento de dados fica prejudicado
      - Novas maneiras de compartilhar áreas de memória e recursos



## Tarefas

- **Processos vs Threads**
  - Todas as tarefas em processo único
    - Cada tarefa é uma thread
      - Compartilham endereços de memória e recursos
      - Implementação mais simples para interação entre tarefas
      - Execução mais ágil
      - Problema: Um erro em uma thread pode se propagar para outras threads
        - » Arquivo corrompido
        - » Entrada inválida...

# Sistemas Operacionais

## Tarefas

- **Processos vs Threads**
  - Sistemas modernos: abordagem híbrida
    - Uma aplicação pode possuir vários processos
    - Cada um contém diversas threads
    - Alia-se a agilidade, isolamento de processos e compartilhamento de dados entre tarefas

## Tarefas

- Exercícios

- 1) Quais as principais vantagens e desvantagens de *threads* em relação a processos?
- 2) Associe as afirmações a seguir aos seguintes modelos de *threads*: a) *many-to-one* (N:1); b) *one-to-one* (1:1); c) *many-to-many* (N:M):
  - a) Tem a implementação mais simples, leve e eficiente.
  - b) Multiplexa os *threads* de usuário em um pool de *threads* de núcleo.
  - c) Pode impor uma carga muito pesada ao núcleo.
  - d) Não permite explorar a presença de várias CPUs pelo mesmo processo.
  - e) Permite uma maior concorrência sem impor muita carga ao núcleo.
  - f) Geralmente implementado por bibliotecas.
  - g) É o modelo implementado no Windows NT e seus sucessores.
  - h) Se um *thread* bloquear, todos os demais têm de esperar por ele.
  - i) Cada *thread* no nível do usuário tem sua correspondente dentro do núcleo.
  - j) É o modelo com implementação mais complexa.