Sistemas Operacionais

Threads

Processos e *threads*

- Vimos o conceito de processo englobando duas características básicas:
  - propriedade de recursos
    - a um processo é alocado um espaço de endereçamento virtual para manter a sua imagem
    - de tempos em tempos o processo pode ter mais memória, além do controle de arquivos, dispositivos de E/S, ...

Processos e *threads* (2)

- unidade de despacho (utilização de CPU):
  - um processo é uma linha de execução
  - esta linha de execução é intercalada com outras linhas de outros processos
  - cada uma delas tem um estado de execução e uma prioridade
  - é a entidade que é escalonada e despachada a CPU pelo SO

Processos e *threads* (3)

- Estas duas características podem ser tratadas de forma independente pelo SO:
  - *thread* ou processo peso leve (*lightweight process*): é a unidade de despacho
  - processo ou tarefa: é a unidade de alocação de recursos

Assim, ...

- Em um ambiente *multithreaded*, um processo:
  - é a unidade de alocação e proteção de recursos
  - tem um espaço de endereçamento virtual que mantém a imagem do processo
  - tem acesso controlado
    - a outros processos,
    - a outros processadores
    - arquivos e
    - outros recursos

e ...

- Em um processo podem existir uma ou mais *threads* com
  - um estado de execução (pronta, ...)
  - seu contexto salvo quando não estiver executando
    - diferentes valores de PC dentro de um processo
  - sua pilha de execução
    - cada *thread pode chamar procedimentos*
  - acesso a variáveis locais próprias
  - acesso compartilhado com outras *threads* deste processo aos recursos do processo

e ...

- um thread, compartilha com outras threads do mesmo processo
  - seção de código
  - outros recursos do SO
  - arquivos abertos
- um processo tradicional (ou pesado) possui uma única thread de controle

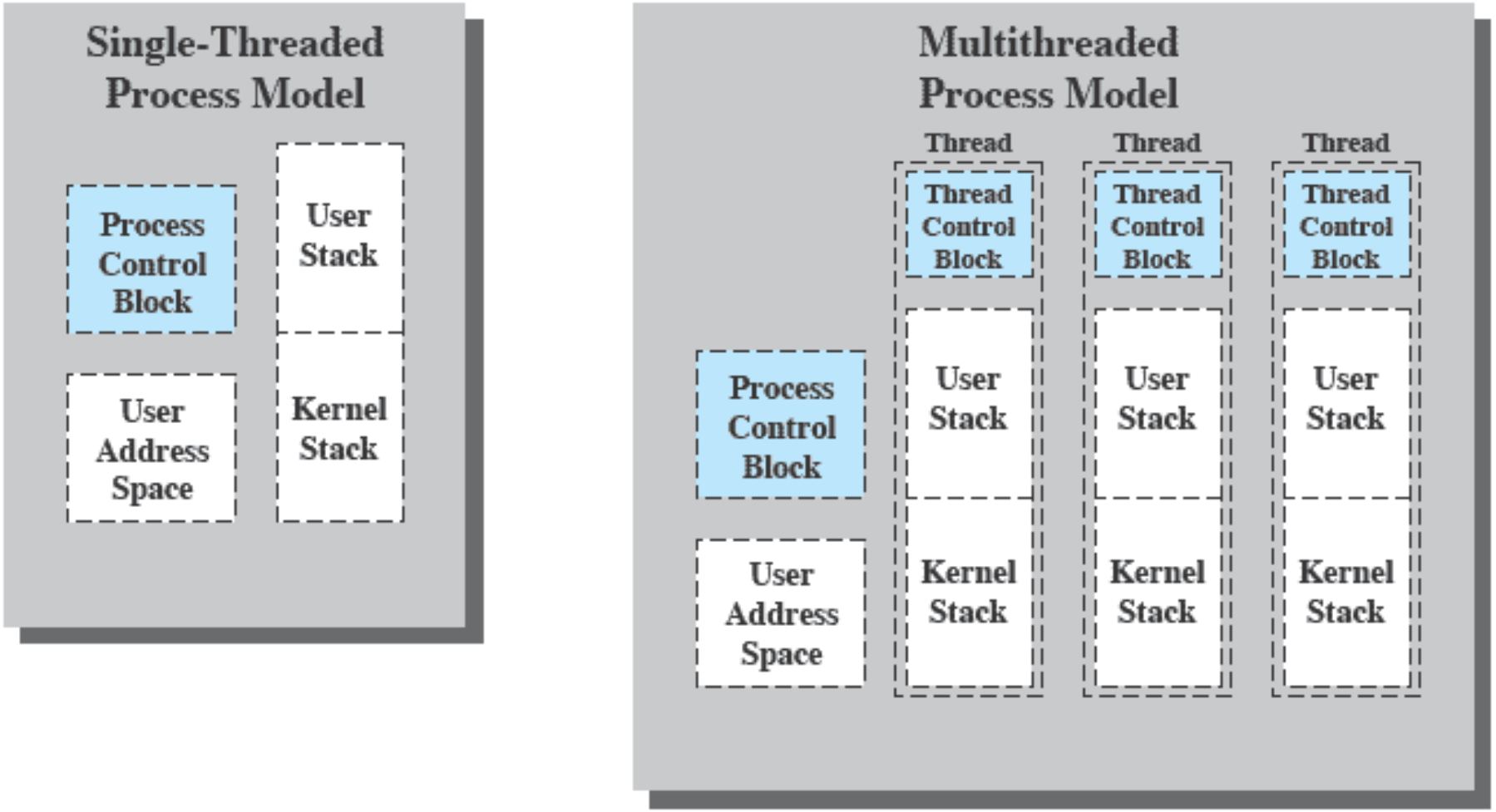


Figure 4.2 Single Threaded and Multithreaded Process Models

Benefícios de *threads*

- É mais rápido criar uma *thread* que um processo
- É mais rápido terminar uma *thread* que um processo
- É mais rápido chavear entre *threads* de um mesmo processo
- *Threads* podem se comunicar sem invocar o núcleo já que compartilham memória e arquivos
  - no caso de comunicação entre processos, a intervenção do núcleo é necessária para proteção e sincronização

Contudo,

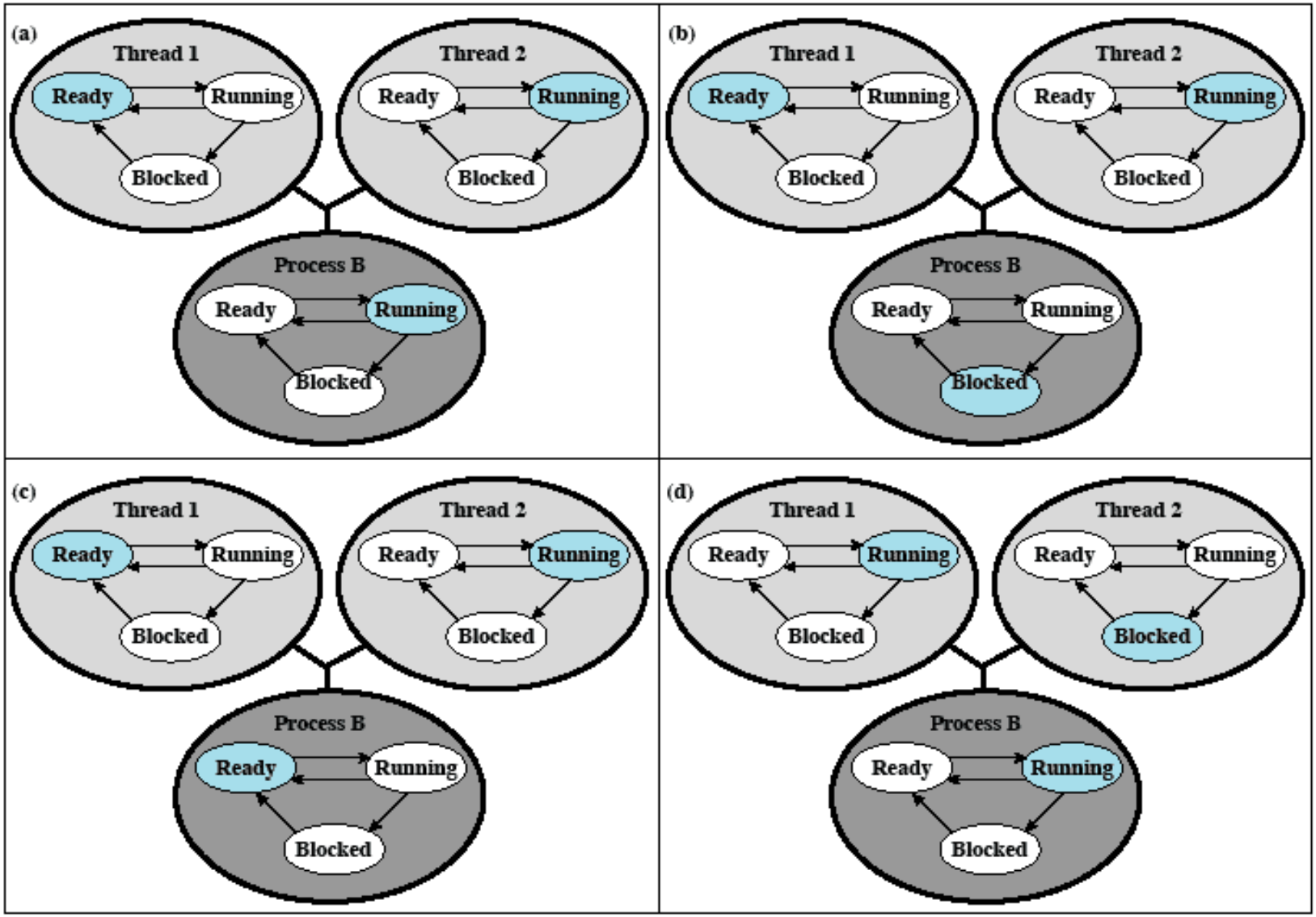
- Suspender um processo implica em suspender todas as *threads* deste processo já que compartilham o mesmo espaço de endereçamento
- O término de um processo implica no término de todas as *threads* desse processo

Estados de uma *thread*

- Estados fundamentais
  - executando, pronto e bloqueado
- Faz sentido o estado "suspensa"?
- O que acontece com as *threads* de um processo quando uma delas bloqueia?

Exemplos de uso de *threads*

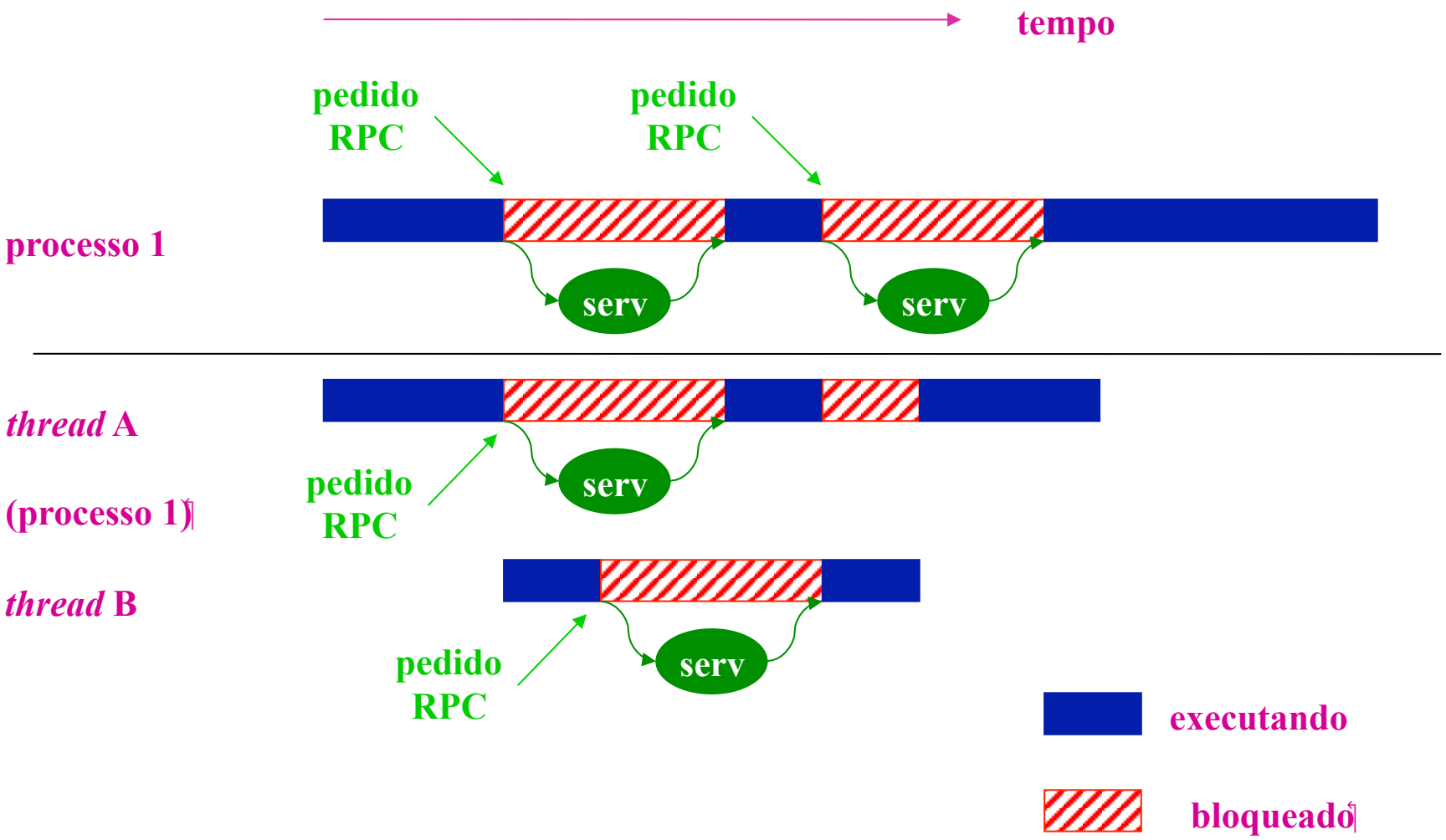
- um editor de rascunho
  - trabalho em primeiro e segundo planos: E/S e cálculo em planilhas
  - *thread 1*: mostra menu e lê entrada
  - *thread 2*: atualização do rascunho
- Processamento assíncrono
  - salvamento periódico em editores de texto
- Aumento de velocidade de execução
  - paralelismo
- Organização
  - facilidade de projeto e implementação



Colored state
is current state

Figure 4.7 Examples of the Relationships Between User-Level Thread States and Process States

Exemplo: RPC


tempo
pedido RPC
pedido RPC
processo 1
serv
serv
thread A
(processo 1)
thread B
pedido RPC
serv
pedido RPC
serv
executando
bloqueado

Operações associadas aos estados das *threads*

- spawn de um processo
  - todas as threads também são criadas
- spawn de uma thread
  - criados somente + região p/ stack + código e dados locais
- bloqueamento/desbloqueamento
  - uma thread fica bloqueada devido ao evento
  - se torna pronta quando o evento ocorre

Em que nível implementar?

Threads no nível do usuário

- gerenciamento das *threads* é feito pela aplicação
  - sem intervenção do SO
- o núcleo desconhece a existência de *threads*
- *bibliotecas para*
  - *criação e destruição de threads*
  - *envio de msgs*
  - *escalonamento de threads*
  - *salvamento e recuperação de contexto*
- chaveamento entre *threads* não requer privilégio de modo núcleo

e mais ...

- implementadas através de bibliotecas: executam em qualquer SO

- Porém:
  - chamada ao sistema bloqueia todas as *threads* de um processo (para o SO é só um processo)
  - não aproveita os benefícios do multiprocessamento (estão em algum processo!)

Em que nível implementar?

Nível do núcleo

- gerenciamento das *threads* é feito pelo núcleo
- núcleo mantém a informação de contexto para processo e *threads*
- escalonamento e chaveamento das *threads* é feito pelo núcleo
- bloqueio de uma *thread* não bloqueia as outras

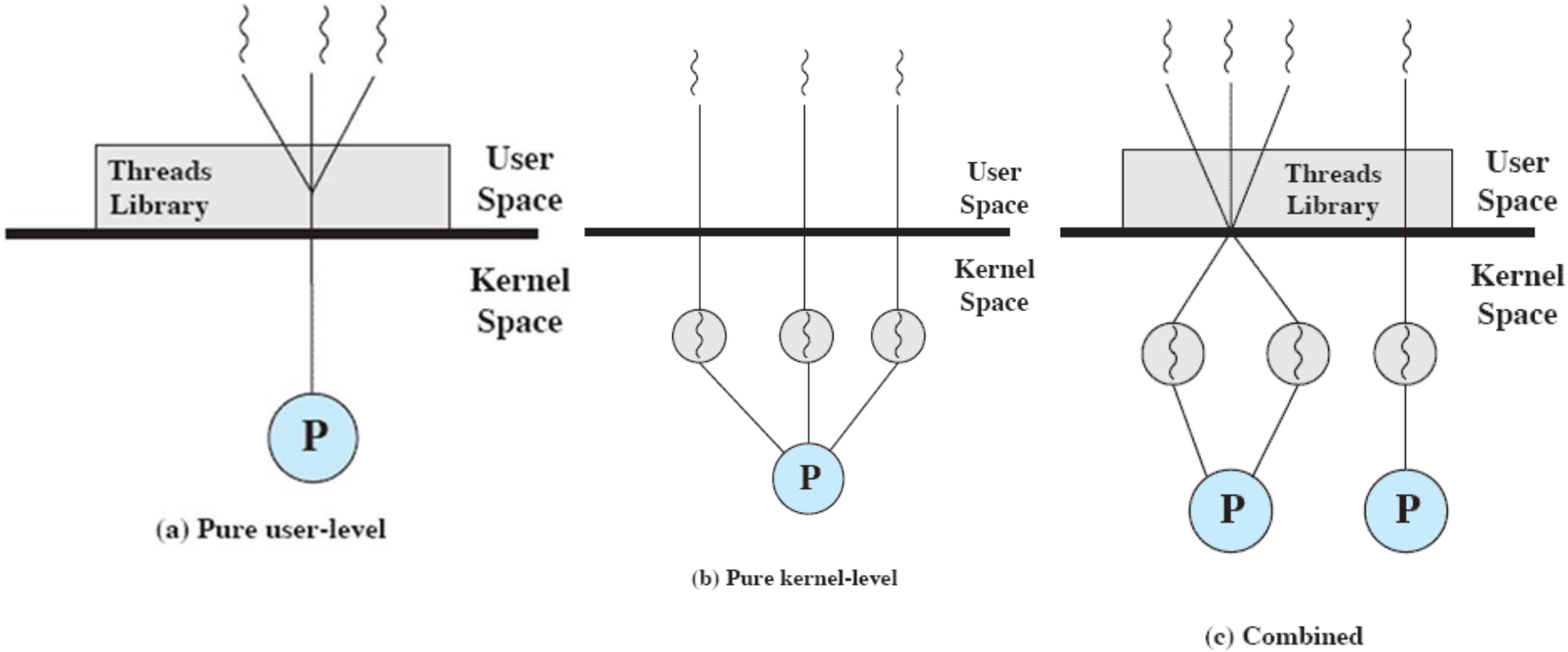
e ainda ...

- *threads* podem aproveitar a capacidade de multiprocessamento
- usuário enxerga uma API para *threads* do núcleo

- Porém:
  - a transferência de controle entre *threads* de um mesmo processo requer chaveamento para modo núcleo
  - chaveamento de threads = troca de contexto (no entanto mais barata que troca de contexto de processos)

Abordagem mista

- Combinar benefícios
- Minimizar desvantagens


Threads
Library
User
Space
Kernel
Space
P
(a) Pure user-level
User
Space
Kernel
Space
P
(b) Pure kernel-level
Threads
Library
User
Space
Kernel
Space
P
P
(c) Combined

Comparando implementações

Latências de operação (μs)

Operação	*Threads*: nível usuário	*Threads*: nível núcleo	Processos
Fork nulo	34	948	11.300
Signal-wait	37	441	1.840

Obs.:

1. VAX monoprocessador executando SO tipo Unix
2. chamada de procedimento neste VAX: ≈ 7μs
3. *trap* ao núcleo: ≈ 17μs

Exemplo de uso: pthreads

- tutorial de *pthreads*

  https://computing.llnl.gov/tutorials/pthreads

- historicamente, vendedores de hardware foram implementando suas próprias versões de threads
  - dificuldade na portabilidade
- Unix especificou essa interface como IEEE POSIX 1003.1c standard (1995) – é o POSIX threads, or Pthreads.
- se tornou padrão entre vendedores, que oferecem pthreads através de seus APIs

Exemplo de uso: pthreads

- Pthreads são definidos como um conjunto de tipos e procedimentos da linguagem C

```
#include  pthread.h
```

Trabalho: exercícios com ptreads

Exercícios com Pthreads

- em

https://computing.llnl.gov/tutorials/pthreads/#CreatingThreads

executar

  - Example 1
  - Example 2
  - Example 3

Threads

- Para entender melhor sobre threads e sua criação:
- Seja um programa composto de um conjunto de procedimentos
  - Imagine que agora, cada procedimento é uma thread que possa ser pelo SO executada simultaneamente e/ou independentemente → é um programa multi-threaded

Threads

- Um exemplo: criação de processos no UNIX
  - Requer uma certa sobrecarga
  - SO tem que guardar informações tais como:
    - Process ID, process group ID, user ID, and group ID
    - Ambiente
    - Working directory.
    - Código
    - Contexto
    - Stack/Heap
    - File descriptors
    - Sinais
    - Bibliotecas compartilhadas
    - Ferramentas de comunicação entre processos (ex.: filas de mensagens, semáforos, etc).

Threads

- Um exemplo: criação de processos no UNIX
  - Requer uma certa sobrecarga
  - SO tem que guardar informações tais como:
    - Process ID, process group ID, user ID, and group ID
    - Ambiente
    - Working directory.
    - Código
    - Contexto
    - Stack/Heap
    - File descriptors
    - Sinais
    - Bibliotecas compartilhadas
    - Ferramentas de comunicação entre processos (ex.: filas de mensagens, semáforos, etc).

Threads

Threads definidas dentro de um processo

- podem utilizar os recursos alocados ao processo
- mas podem ser escalonados pelo SO para executarem de forma independente
  - Os recursos necessários para que sejam executados de forma independente são duplicados
  - O fluxo independente é possível pois cada thread tem alocado a si
    - Stack pointer
    - Registradores (contexto)
    - Propriedades associadas a escalonamento
    - Conjunto de sinais bloqueados
    - Dados específicos aquela thread

Threads

- Atenção: devido a possibilidade de compartilhamento de recursos do processo em questão:
  - Se uma thread modifica algum recurso compartilhado as outras threads que participam do compartilhamento presenciam a mudança
    - (ex.: um fechamento de um arquivo compartilhado)
  - Leitura e escrita em um mesmo arquivo é possível – necessário explicitar sincronização

Por que *Threads*

Entidade mais leves do que processos

- Custo de criação e gerenciamento de threads é menor do que de processos
  - Espaço de pilha e TCB (para o contexto)
- Threads requerem menos recursos de gerenciamento do SO do que processos
  - As threads não necessitam de um novo espaço de armazenamento, por exemplo, como processos podem vir a precisar
- Possibilidade de sobrepor trabalho realizado em CPU com E/S
  - Mas as threads para tal devem ser definidas