# Fundamentos de Sistemas Operacionais

**Professor: Cristiano Bonato Both**

# Sumário

- Processos leves
  - Relembrando o funcionamento de processos pesados
  - Implementação do conceito de processos e threads
    - Modelo N:1
    - Modelo 1:1
    - Modelo M:N
- Referência

# Relembrando a aula passada

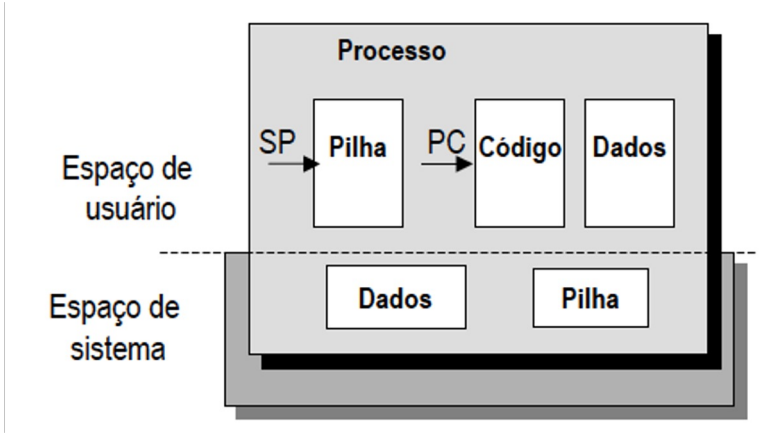
- Multiprogramação pressupõe a existência simultânea de vários processos disputando o processador
- Necessidade de “intermediar” esta disputa de forma justa
  - Gerência do processador
    - Algoritmos de escalonamento
- Necessidade de “representar” um processo
  - Implementação de processos
    - Estruturas de dados

# Relembrando a aula passada

- Processo é um programa em execução
  - Áreas na memória para código, dados e pilha
- Possui uma série de estados (apto, executando, bloqueado, etc.) para representar sua evolução no tempo, implicando em:
  - Organizar os processos nos diferentes estados
  - Determinar quando um processo tem direito a “utilizar” o processador
- Necessário manter informações a respeito do processo
  - *e.g.*, prioridades, localização em memória, estado atual, direitos de acesso, recursos que emprega, etc.

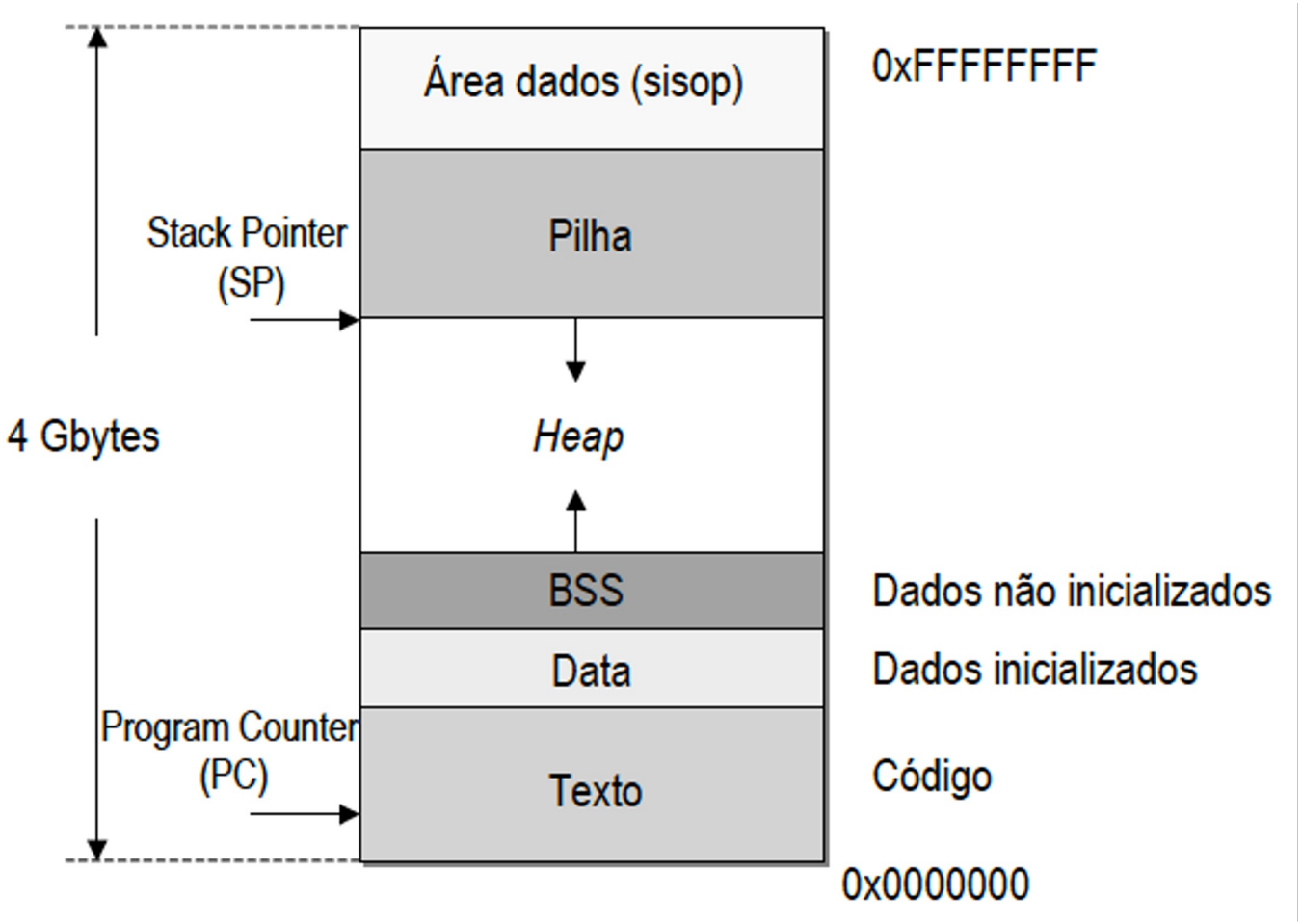
# Relembrando a aula passada

- Processo é representado por:
  - Espaço de endereçamento: área para armazenamento da imagem do processo
  - Estruturas internas do sistema (tabelas internas, etc.)
    - Mantidos no descritor de processos
  - Contexto de execução (pilha, programa, dados, etc.)



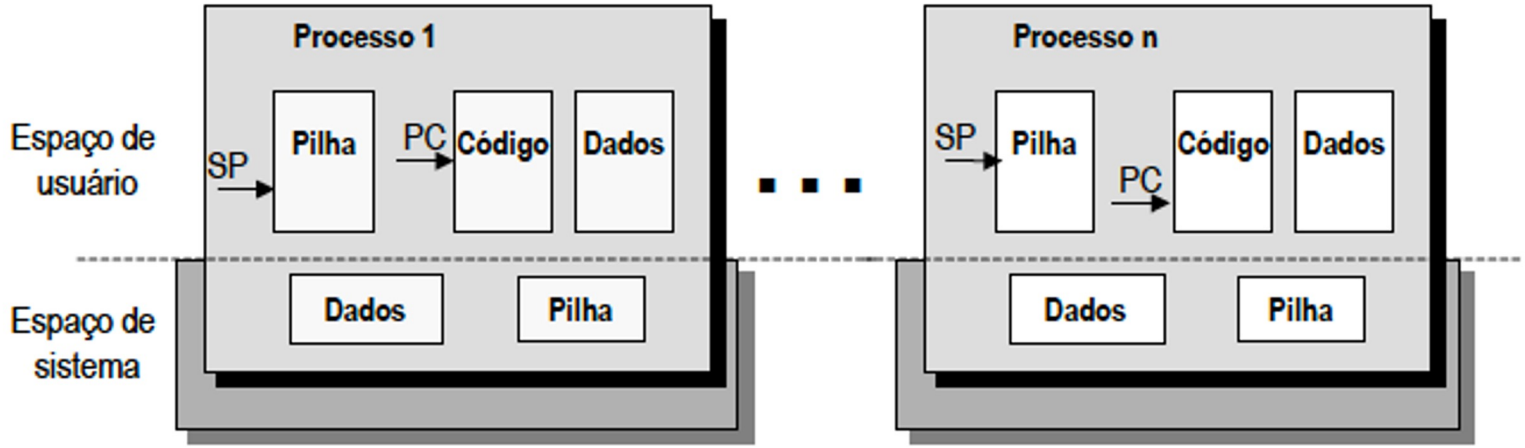

PC = *Program Counter*
SP = *Stack Pointer*

# Relembrando a aula passada



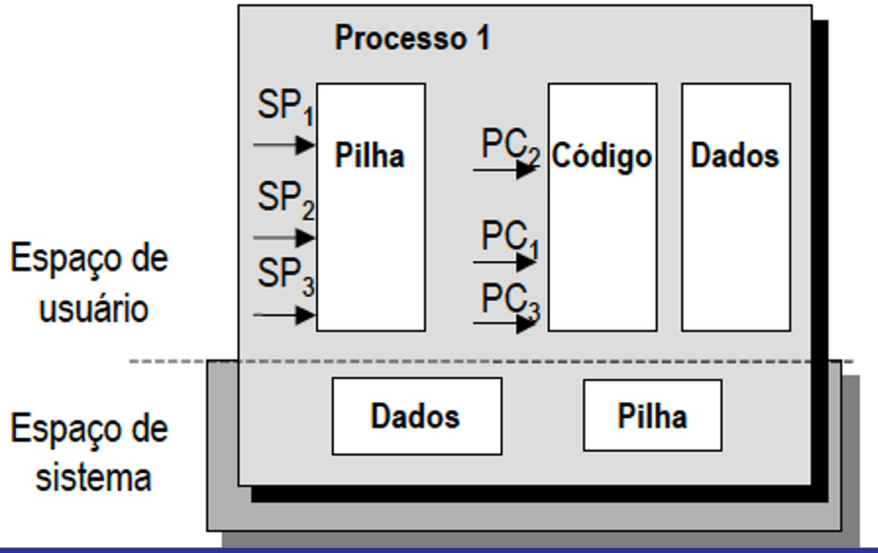
# Vários processos

- Um fluxo de controle por processo (*thread*)
- Troca de processo implica em atualizar estruturas de dados internas do sistema operacional
  - *e.g.*, contexto, espaço de endereçamento, etc.



# Vários fluxos em um único processo

- Um fluxo de instrução é implementado através do Contador de Programa (PC) e de uma pilha (SP)
- Estruturas comuns compartilhadas
  - Código
  - Dados
  - Descritor de processo
- Conceito de *threads*



# Multiprogramação pesada

- Custos de gerenciamento do modelo de processos
  - Criação de processo
  - Troca de contextos
  - Esquemas de proteção, memória virtual, etc.
- Custos são fatores limitantes na interação de processos
  - Mecanismos de IPC (*Inter Process Communication*) necessitam tratamento de estruturas complexas que representam o processo e suas propriedades
- Solução
  - "Aliviar" os custos, *i.e.*, reduzir o "peso" das estruturas envolvidas

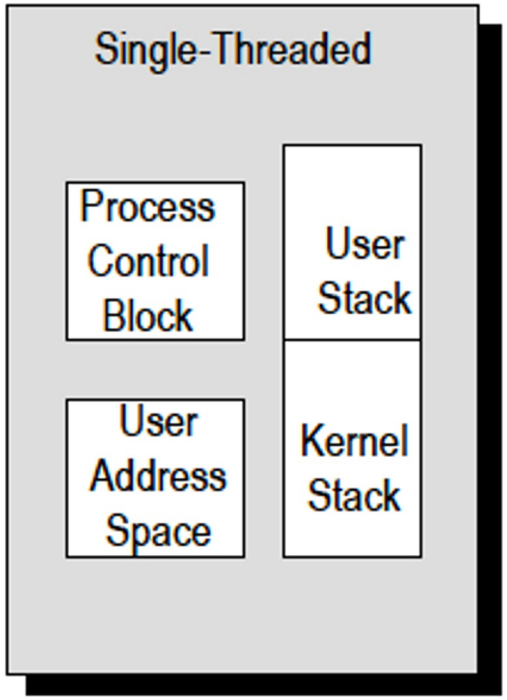
# Multiprogramação leve

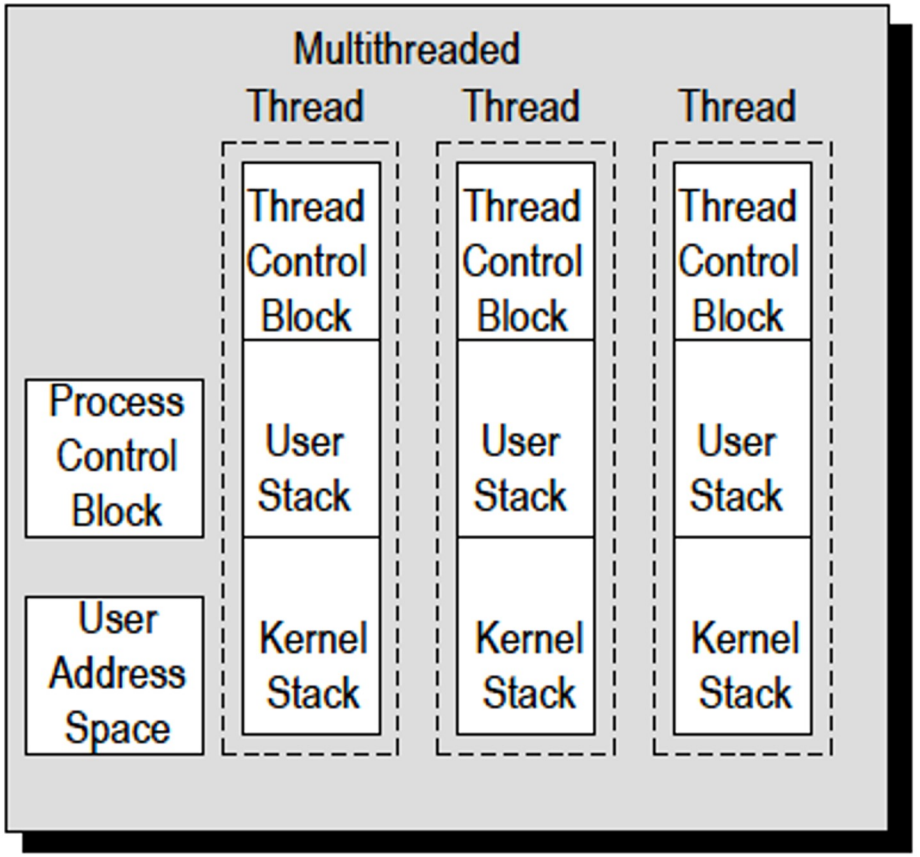
- Fornecido pela abstração de um fluxo de execução (*thread*)
- Unidade de interação passa a ser uma função
- Contexto de uma *thread*
  - Registradores (pilha, apontador de programa, registradores de uso geral)
- Comunicação através do compartilhamento direto da área de dados

# Implementação de *threads*

- *Threads* são implementadas através de estruturas de dados similares ao descritor de processo
  - Descritor de *threads*
  - Menos complexa (mais leve)
- Podem ser implementadas em dois níveis diferentes:
  - Espaço de usuário
  - Espaço de sistema

# Modelos de processos *Single Threaded* e *Multithreaded*



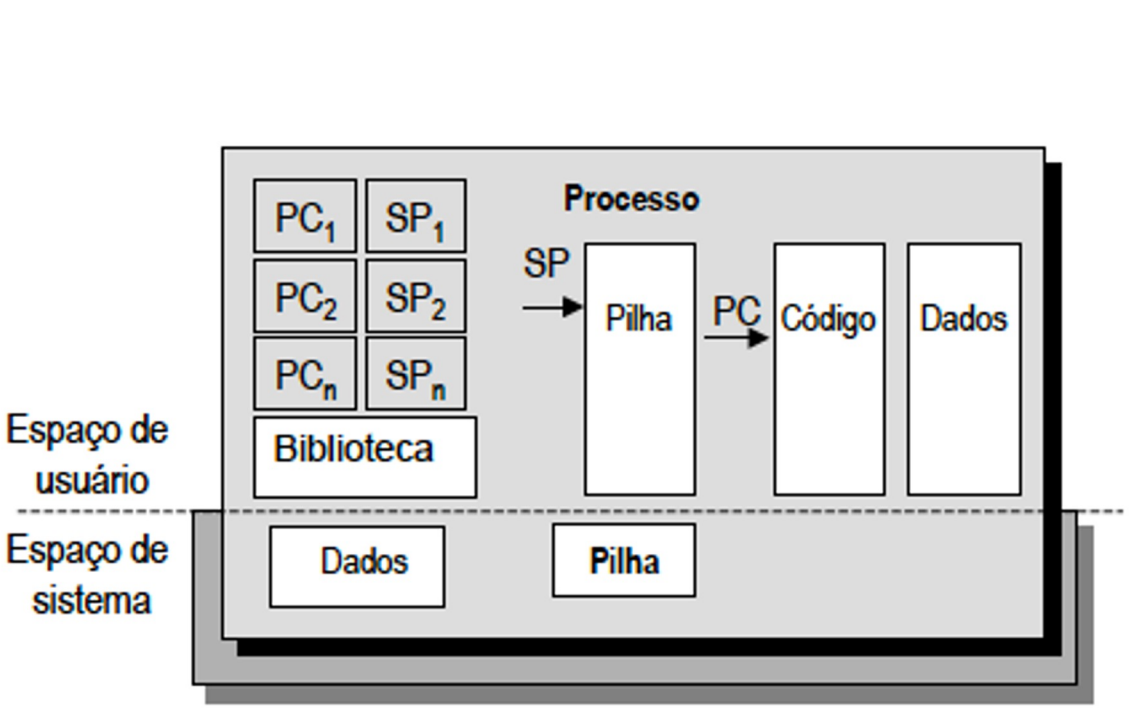



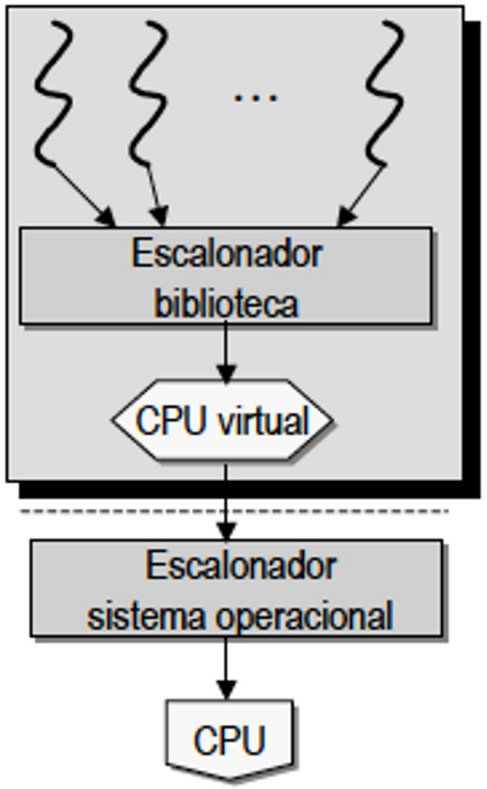

https://www.youtube.com/watch?v=_5q8ZK6hwzM

# Modelo N:1

- *Threads* em nível de usuário
  - *User-level threads* ou ainda *process scope*
- Todas as tarefas de gerenciamento de *threads* são feitas em nível da aplicação
  - *Threads* são implementadas por uma biblioteca que é ligada ao programa
  - Interface de programação (API) para funções relacionadas com *threads*
    - *e.g.*, criação, sincronismo, término, etc.
- O sistema operacional não “enxerga” a presença das *threads*
- A troca de contexto entre *threads* é feita em modo usuário pelo escalonador embutido na biblioteca
  - Não necessita privilégio especiais
  - Escalonamento depende da implementação

# Implementação do modelo N:1



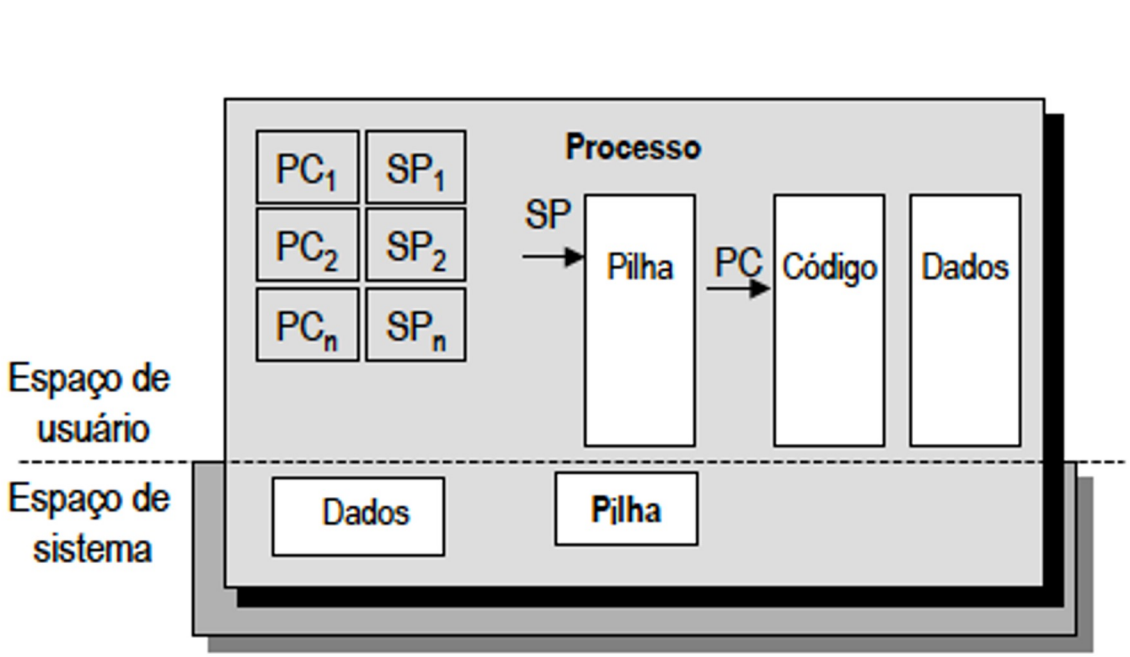
# Vantagens e desvantagens

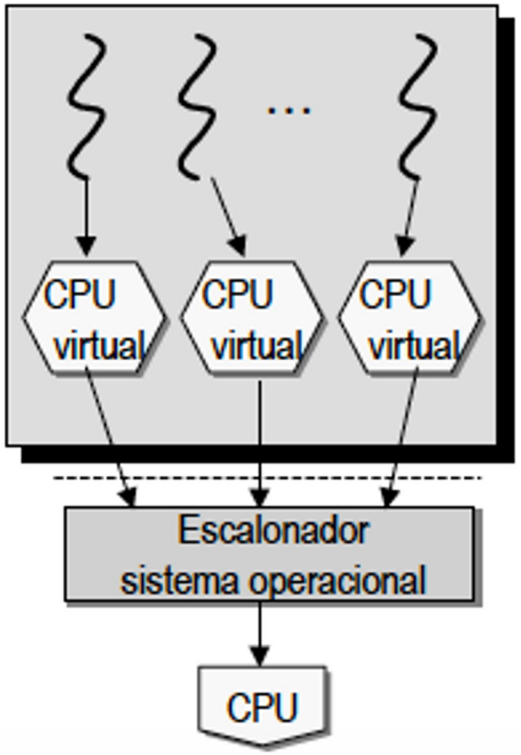
- Vantagens:
  - Sistema Operacional divide o tempo do processador entre os processos “pesados” e a biblioteca de *threads* divide o tempo do processo entre as *threads*
  - Leve: sem interação/intervenção do Sistema Operacional
- Desvantagens:
  - Uma *thread* que realiza uma chamada de sistema bloqueante causa o bloqueio de todo o processo
    - *e.g.*, operações de entrada/saída
  - Não explora paralelismo em máquinas multiprocessadas

# Modelo 1:1

- *Threads* em nível do sistema
  - *Kernel level threads* ou ainda *system scope*
- Resolver desvantagens do modelo N:1
- O Sistema Operacional "enxerga" as *threads*
  - Sistema operacional mantém informações sobre processos e sobre *threads*
  - Troca de contexto necessita a intervenção do Sistema Operacional
- O conceito de *threads* é considerado na implementação do Sistema Operacional

# Implementação do modelo 1:1



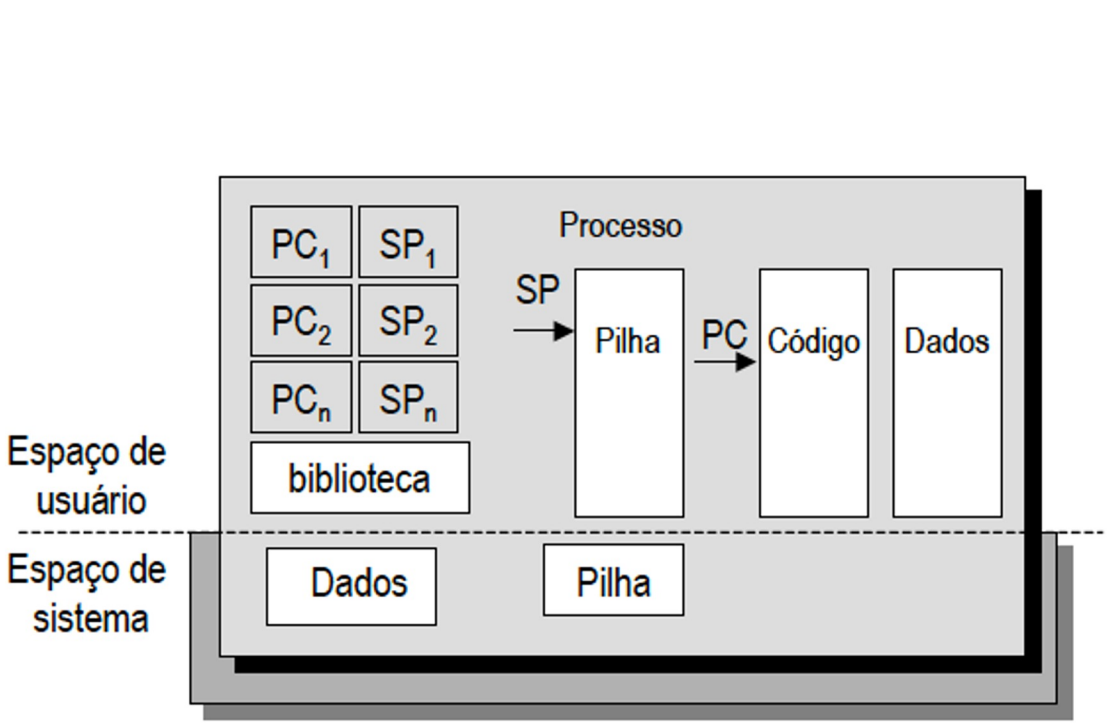
# Vantagens e desvantagens

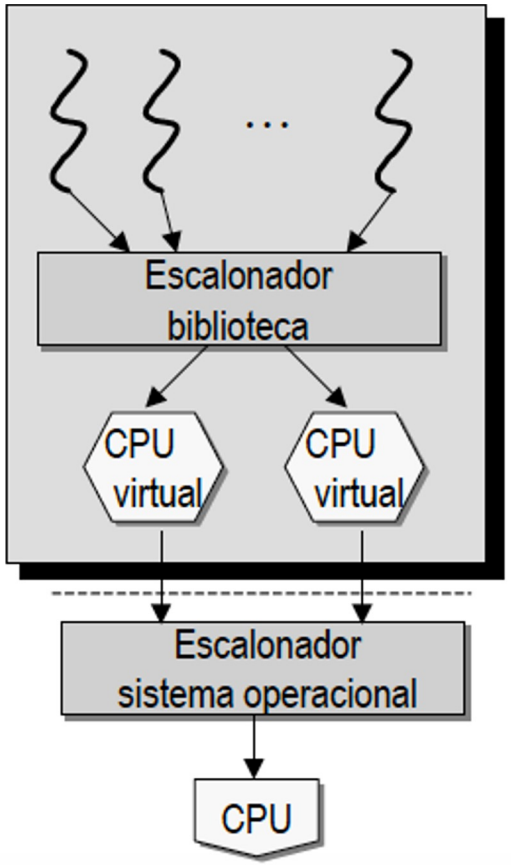
- Vantagens:
  - Explora o paralelismo de máquinas multiprocessadoras (SMP)
  - Facilita o descobrimento de operações de entrada/saída por cálculos
- Desvantagens:
  - Implementação “mais pesada” que o modelo N:1

# Modelo M:N

- Abordagem que combina os modelos N:1 e 1:1
- Oferece dois níveis de escalonamento
  - Nível usuário: *thread* sobre unidade de escalonamento
  - Nível sistema: unidades de escalonamento sobre processador
- Dificuldade é parametrizar M e N

# Implementação do modelo M:N



# Por que utilizar *threads*?

- Permitir a exploração do paralelismo real oferecido por máquinas multiprocessadoras (modelo M:N ou 1:1)
- Aumentar número de atividades executadas por unidade de tempo (*throughput*)
- Diminuir tempo de resposta
  - Possibilidade de associar *threads* a dispositivos de entrada/saída
- Sobrepor operações de cálculo com operações de entrada e saída

# Vantagens de *multithreading*

- Tempo de criação/destruição de *threads* é inferior que tempo de criação/destruição de um processo
- Chaveamento de contexto entre *threads* é mais rápido que tempo de chaveamento entre processos
- Como *threads* compartilham o descritor do processo, elas dividem o mesmo espaço de endereçamento, o que permite a comunicação por memória compartilhada sem interação com o núcleo

# Exemplo

```c
#include <stdio.h>

unsigned long somaNumeros() {
    int i = 0;
    unsigned long soma = 0;

    while(i < 100000) {
        soma += i;
        i++;
    }

    return soma;
}

int main() {
    unsigned long soma;
    soma = somaNumeros();
    printf("%lu\n", soma);

    return 0;
}
```

Programa sequencial que soma os números entre 0 e 100.000

Para compilar:
gcc -o programa programa.c

# Exemplo

- Programa com 4 *threads* que soma os números entre 0 e 100.000
- Cada *thread* soma 25.000 valores

- A *main* faz a soma final

Para compilar:
gcc programa.c -o programa -lpthread

```c
#include <pthread.h>
#include <stdio.h>

unsigned long soma[4];

void* thread_fn(void* arg) {
    long id = (long) arg;
    int inicio = id * 25000;
    int i = 0;

    while(i < 25000) {
        soma[id] += (i + inicio);
        i++;
    }

    return NULL;
}

int main(void) {
    pthread_t t1, t2, t3, t4;
    unsigned long resultado = 0;

    pthread_create(&t1, NULL, thread_fn, (void *)0);
    pthread_create(&t2, NULL, thread_fn, (void *)1);
    pthread_create(&t3, NULL, thread_fn, (void *)2);
    pthread_create(&t4, NULL, thread_fn, (void *)3);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    pthread_join(t3, NULL);
    pthread_join(t4, NULL);

    resultado = soma[0] + soma[1] + soma[2] + soma[3];

    printf("%lu\n", resultado);

    return 0;
}
```

# Exemplo

- Para o código sequencial e o código com threads

| Tempos de Execução (segundos) | | | | |
|---|---|---|---|---|
| | **Sequencial** | **Threads - 2** | **Threads - 4** | **Threads - 5** |
| **100000** | 0.003 | 0.004 | 0.004 | 0.004 |
| **1000000** | 0.008 | 0.01 | 0.01 | 0.01 |
| **10000000** | 0.061 | 0.035 | 0.034 | 0.04 |
| **100000000** | 0.275 | 0.256 | 0.369 | 0.29 |
| **1000000000** | 2.364 | 2.783 | 3.83 | 2.865 |
| **10000000000** | > 30 min | > 30 min | > 30 min | 35.616 |

**Nota:** dependendo do tamanho do problema, o uso de *threads* pode aumentar o tempo de execução em comparação com o código sequencial

# Comparação

| Platform | fork() | | | pthread_create() | | |
|---|---|---|---|---|---|---|
| | real | user | sys | real | user | sys |
| Intel 2.6 GHz Xeon E5-2670 (16 cores/node) | 8.1 | 0.1 | 2.9 | 0.9 | 0.2 | 0.3 |
| Intel 2.8 GHz Xeon 5660 (12 cores/node) | 4.4 | 0.4 | 4.3 | 0.7 | 0.2 | 0.5 |
| AMD 2.3 GHz Opteron (16 cores/node) | 12.5 | 1.0 | 12.5 | 1.2 | 0.2 | 1.3 |
| AMD 2.4 GHz Opteron (8 cores/node) | 17.6 | 2.2 | 15.7 | 1.4 | 0.3 | 1.3 |
| IBM 4.0 GHz POWER6 (8 cpus/node) | 9.5 | 0.6 | 8.8 | 1.6 | 0.1 | 0.4 |
| IBM 1.9 GHz POWER5 p5-575 (8 cpus/node) | 64.2 | 30.7 | 27.6 | 1.7 | 0.6 | 1.1 |
| IBM 1.5 GHz POWER4 (8 cpus/node) | 104.5 | 48.6 | 47.2 | 2.1 | 1.0 | 1.5 |
| INTEL 2.4 GHz Xeon (2 cpus/node) | 54.9 | 1.5 | 20.8 | 1.6 | 0.7 | 0.9 |
| INTEL 1.4 GHz Itanium2 (4 cpus/node) | 54.5 | 1.1 | 22.2 | 2.0 | 1.2 | 0.6 |

Source fork_vs_thread.txt

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

# Referências Bibliográficas


- SILBERSCHATZ, A.; GALVIN, Peter; GAGNE Greg, Operating System Concepts Essentials. John Wiley & Sons, Inc. 2th edition, 2013.
- TANENBAUM, Andrew S. Sistemas operacionais modernos. 3a. ed. São Paulo: Pearson, 2009-2013. p. 653.
- OLIVEIRA, Rômulo; CARÍSSIMI, Alexandre; TOSCANI, Simão. Sistemas Operacionais. Porto Alegre: Bookman, 4a. ed. 2010.