DCA-0125 Sistemas de Tempo Real

Luiz Affonso Guedes
www.dca.ufrn.br/~affonso
affonso@dca.ufrn.br

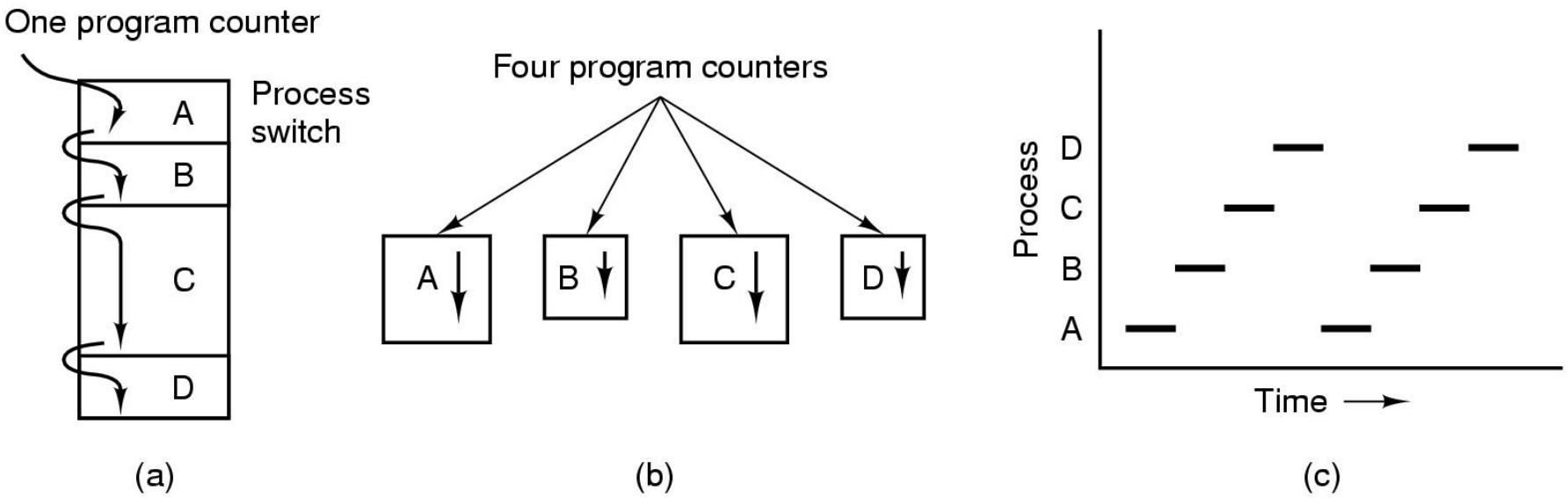
Conceitos de Threads

Conteúdo

- Definição de Threads
- Gerência de Threads
  - Criação, uso, comunicação e eliminação.
- Exemplos da Pthread Posix em C/C++

Multiprogramação

- Tornar mais eficiente o aproveitamento dos recursos do computador.
- Execução "simultânea" de vários programas.
  - Diversos programas são mantidos na memória.
- *O próprio SO é um programa!*



Conseqüências da Multiprogramação

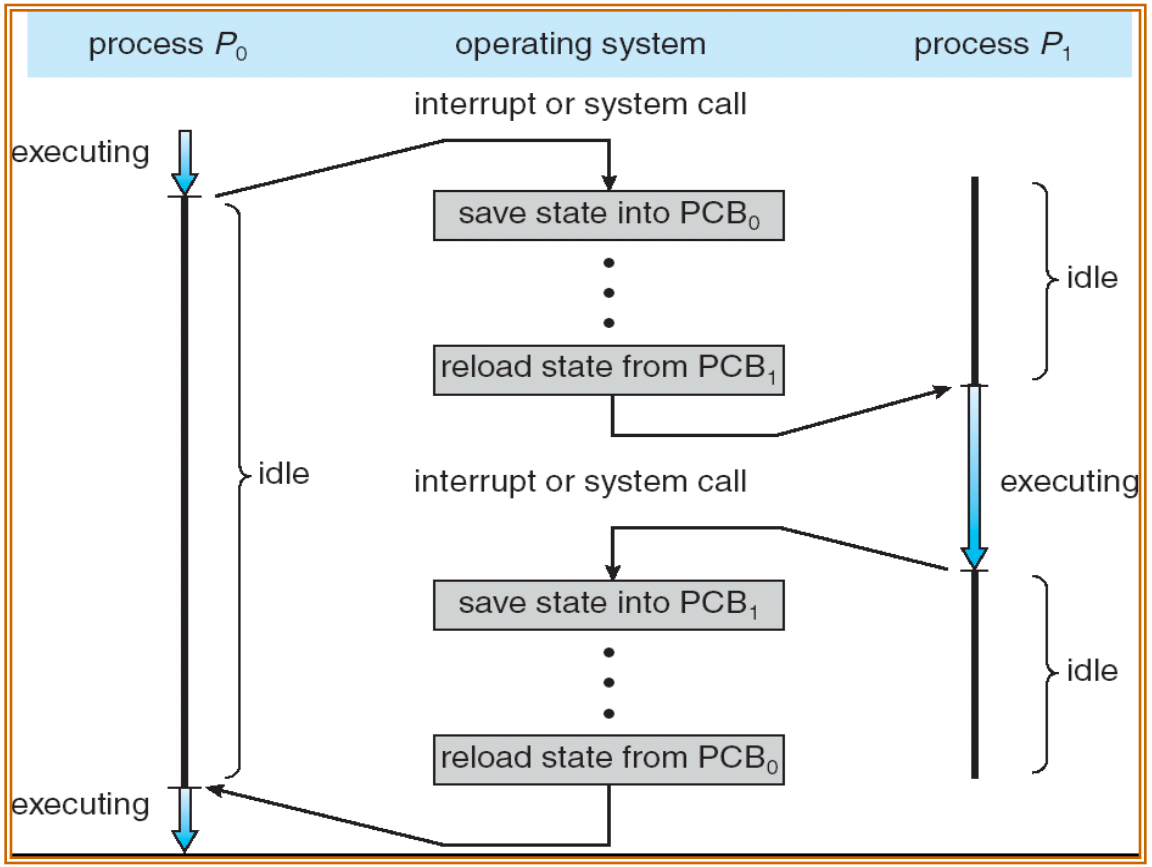
- Necessidade de controle e sincronização dos diversos programas.
- Necessidade de se criar conceitos e abstração novas
  - Modelagem
  - Implementação

Definição de Processo

- Programa
  - Estrutura estática
    - Instruções + Dados
- Processo
  - Entidade Ativa
  - Instância de um Programa em execução.
  - Processos = Programa + Identificador + Entrada + Saída + Estado
  - Dois Processos podem executar instâncias diferentes do mesmo Programa.

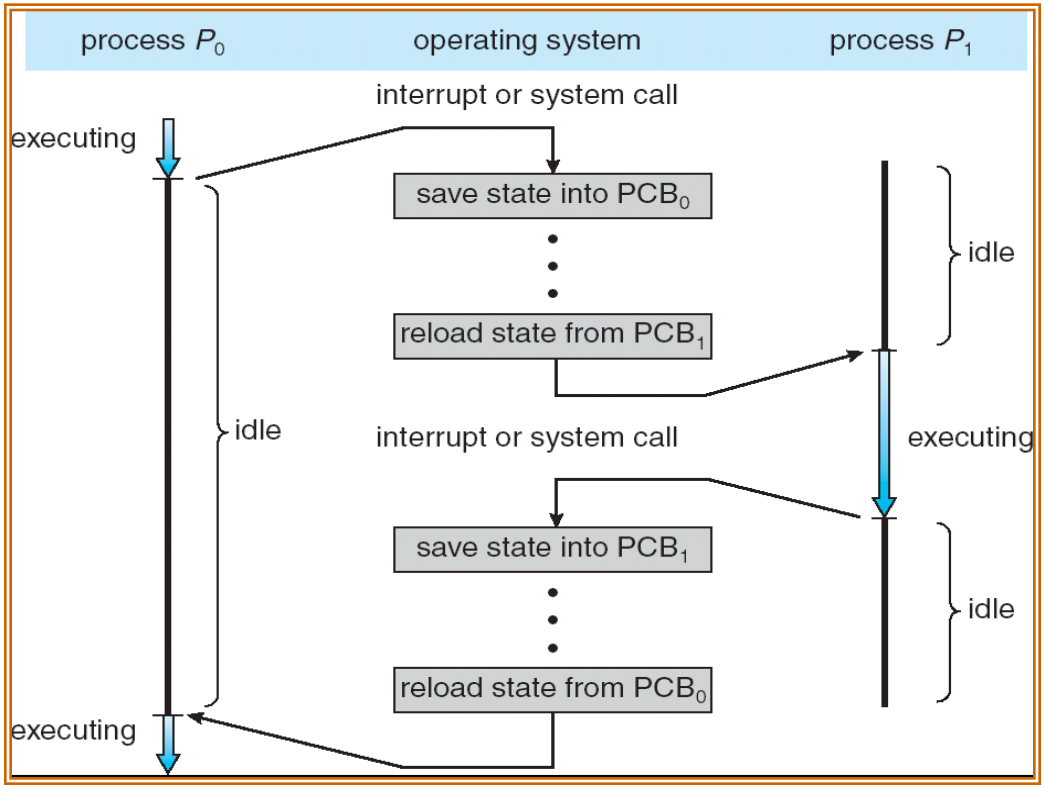
Definição de Processo

- Processo é um programa em execução!
- O SO trata com processos e não com programas.

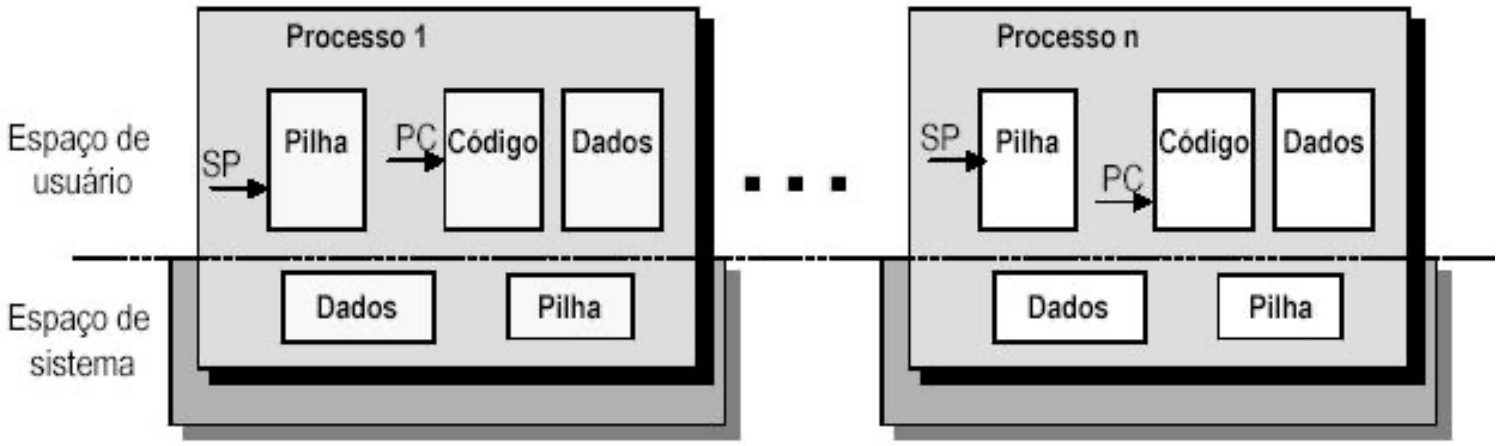


Eficiência no Uso de Processos

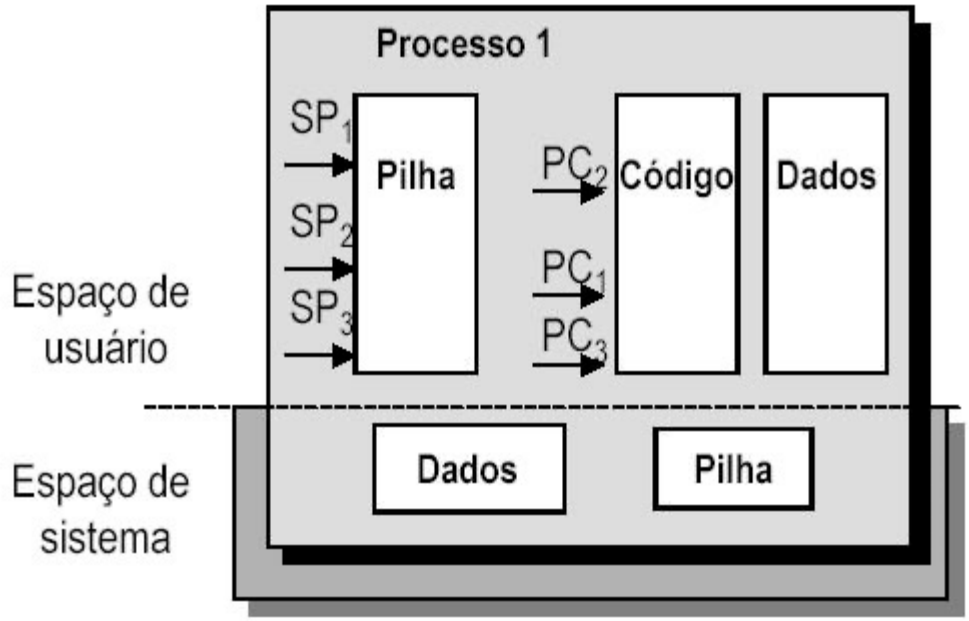
- Overhead de criação de processos
- Overhead de mudança de contexto
- Comunicação entre processos (IPC) mais complexa



Processos com Fluxos Simples

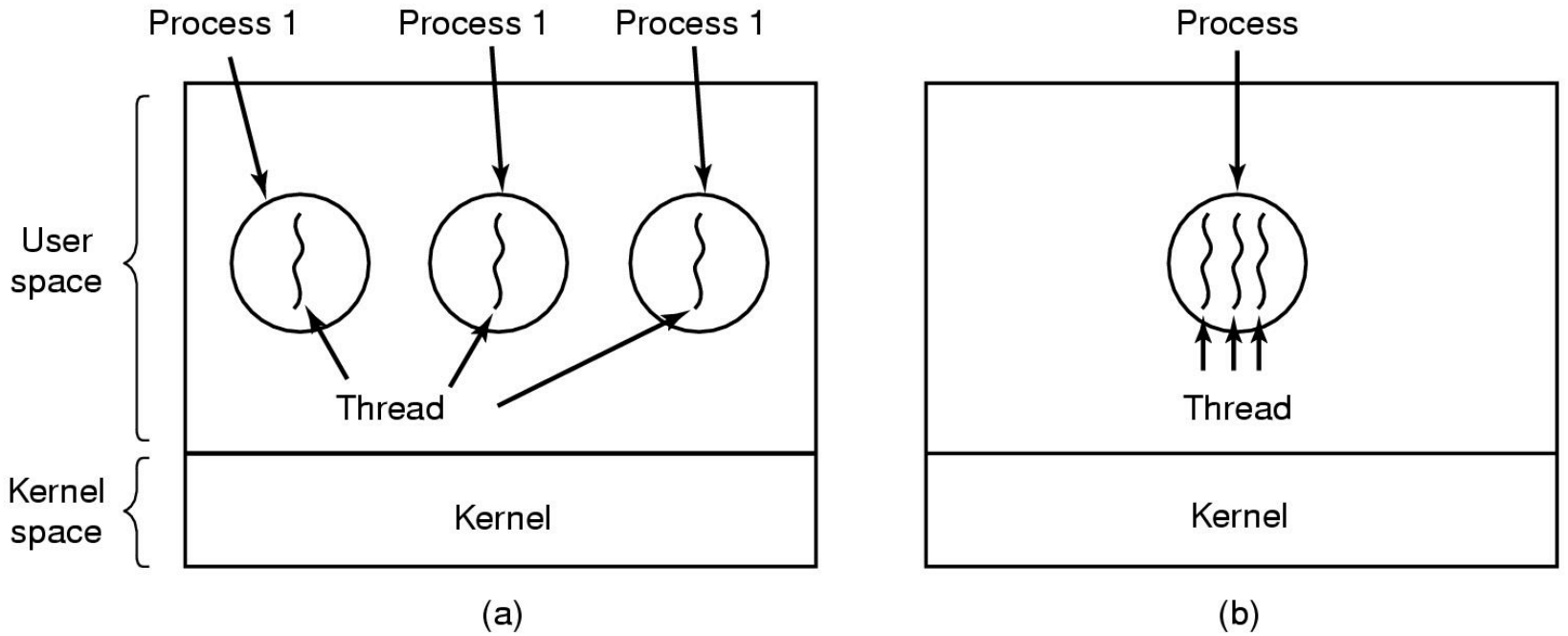


Processos com Múltiplos Fluxos



- Utilizar vários fluxos (threads) num mesmo processo.
- Manter a abstração de Processo
- Diminuir overhead de gerência e mudança de contexto

Threads: Idéia Básica

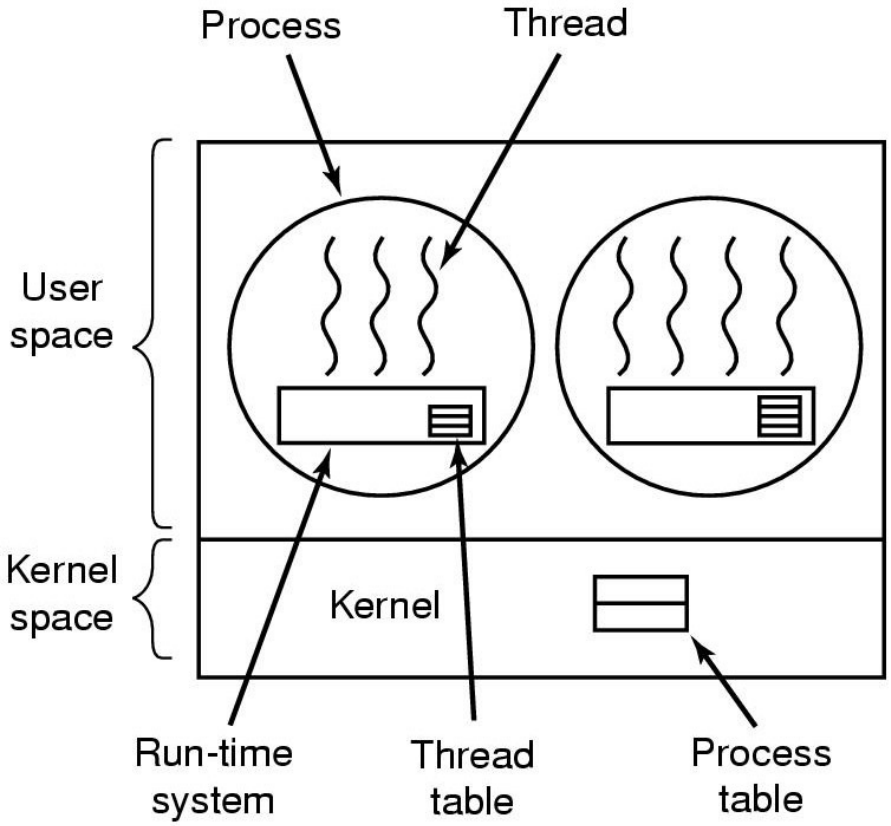


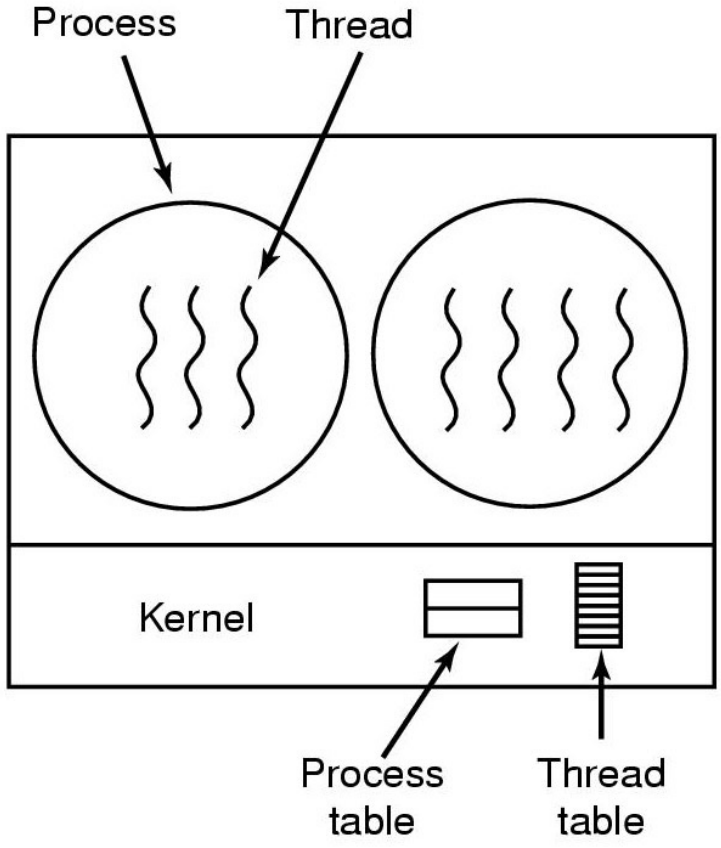
Threads como Processos Leves

- Utilizar vários fluxos (threads) num mesmo processo.
- Aliviar a carga de criação, gerência e mudança de contexto.
- Conceito de Thread
  - Pilha + PC + Registradores de uso geral
  - Abstração similar a Processos
  - Unidade de execução passa ser uma função
  - Comunicação via variáveis globais

Implementação de Threads

- Implementação no nível do usuário
- Implementação no núcleo

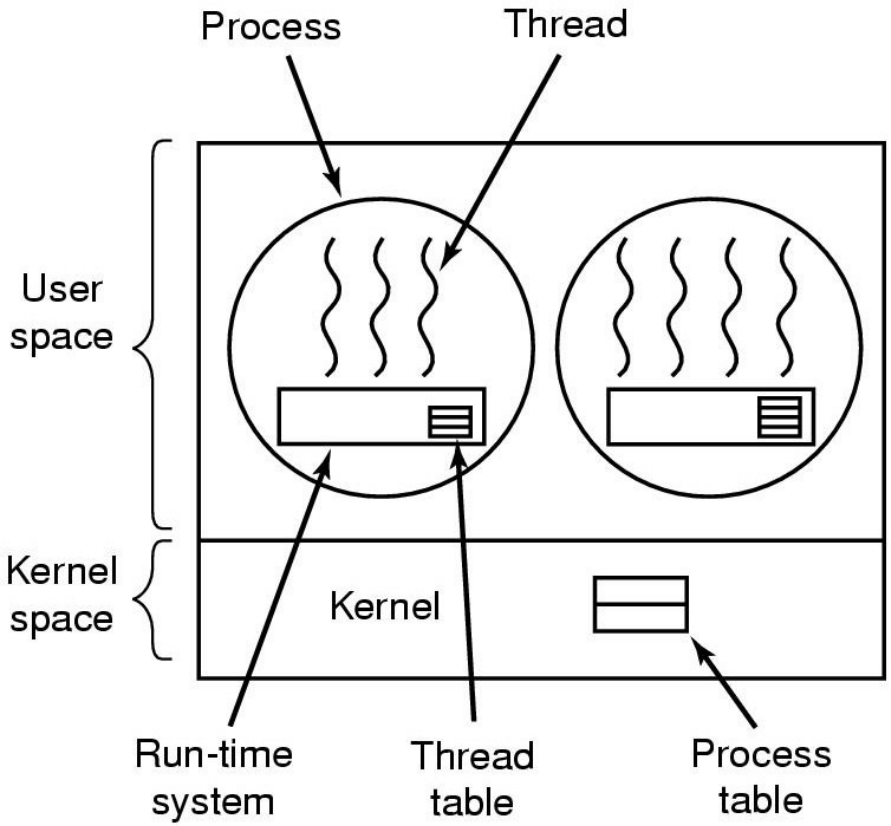




Thread no Nível do Usuário

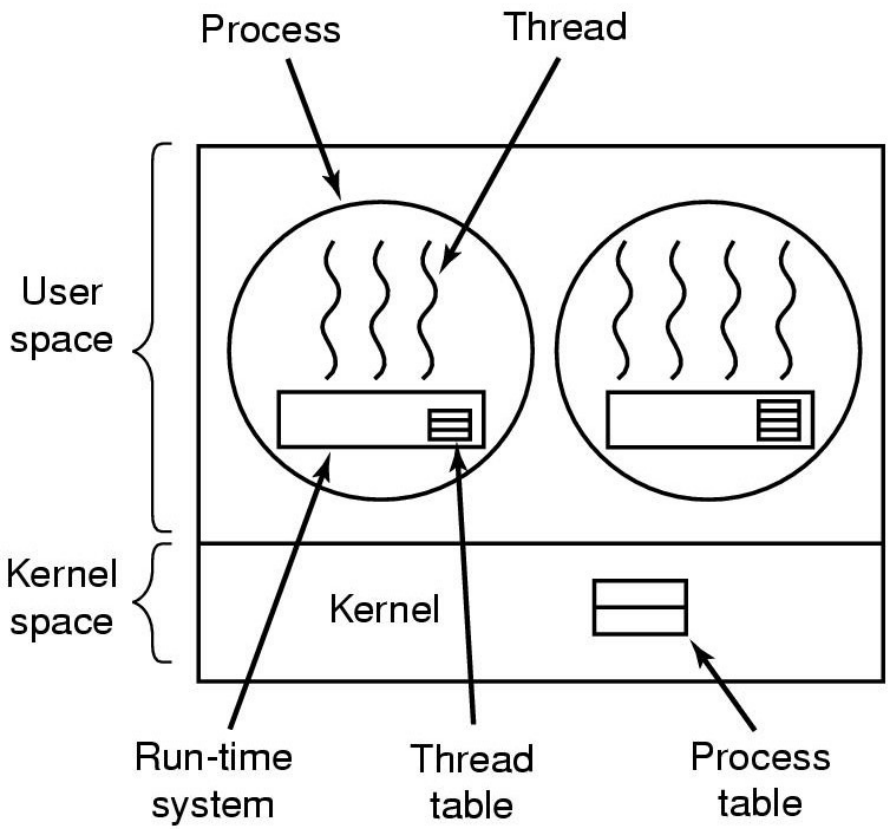
Modelo N:1

- Todo gerenciamento é feito no nível de aplicação
  - Implementada como biblioteca
  - O SO gerencia os Processos apenas, as threads são gerenciadas pela aplicação
  - A mudança de contexto é gerenciada no nível de usuário.
  - O algoritmo de escalonamento é implementado no nível de aplicação.



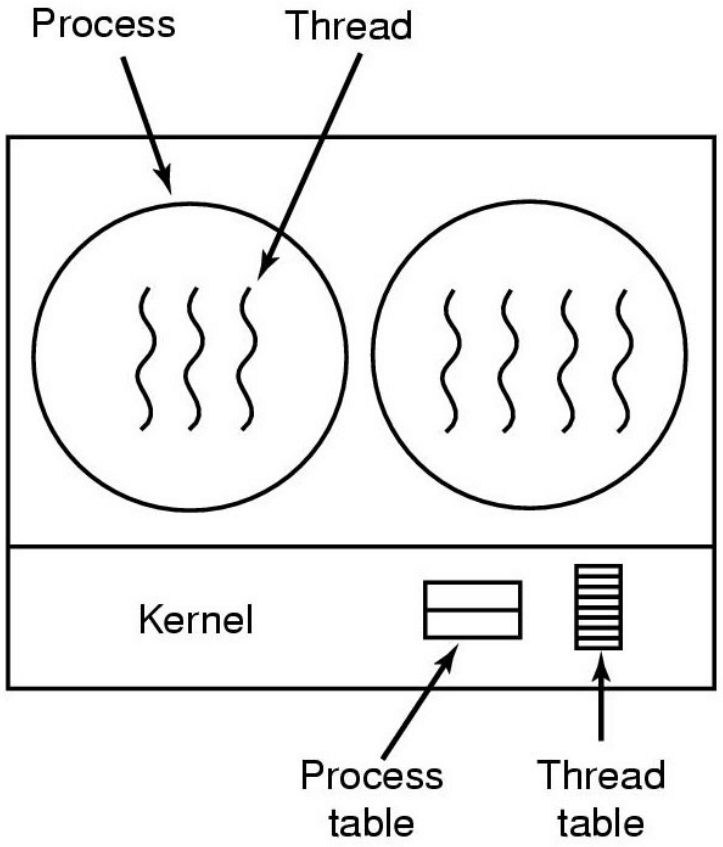
Thread no Nível do Usuário

- Vantagens
  - Criação e gerência mais leve, pois não há a necessidade de se acessar o núcleo.
  - O SO cuida dos Processos e a Biblioteca das Threads
  - Possibilita multithreading em SO sem este suporte
- Desvantagens
  - Uma Thread pode bloquea todo o seu Processo.
  - Não explora paralelismo em máquinas com vários processadores.



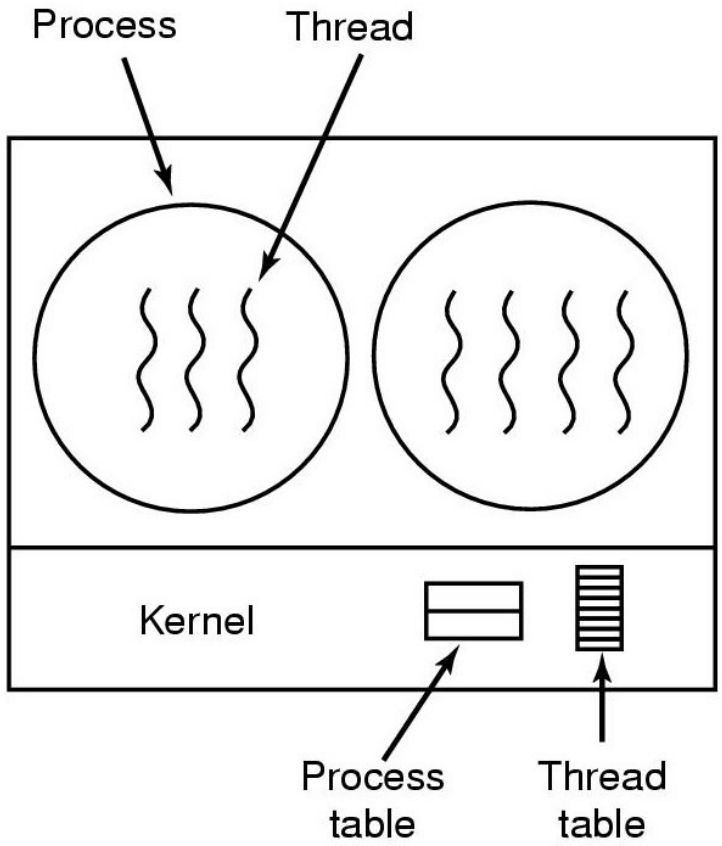
Thread no Nível do Sistema

- Modelo 1:1
  - Implementação no núcleo do SO
- O SO gerencia os Processos e as Threads
  - Troca de contexto entre Threads é efetuado pelo SO.
  - O SO é quem faz o escalonamento de Processos e Threads
- O conceito de threads é incorporado no projeto e implementação do SO



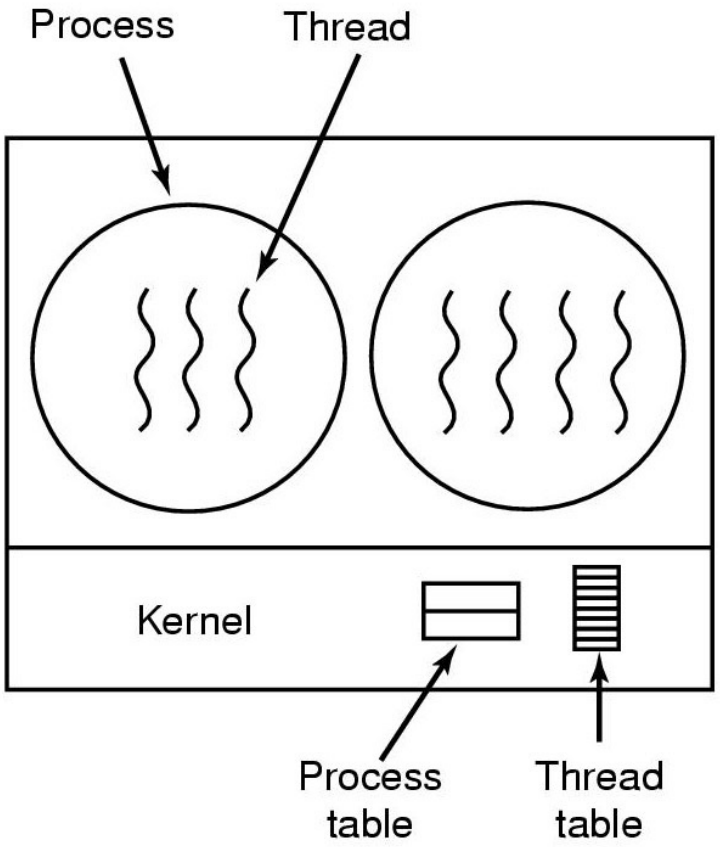
Thread no Nível do Sistema

- Vantagens
  - Permite paralelismo real.
  - Evita possíveis bloqueios desnecessários de threads.
    - Pois o controle é do núcleo e não da aplicação
  - Aplicações menos dependentes de implementações específicas de bibliotecas.
- Desvantagens
  - Maior overhead na criação e gerência de threads, pois essas operações requerem acesso ao núcleo do SO.



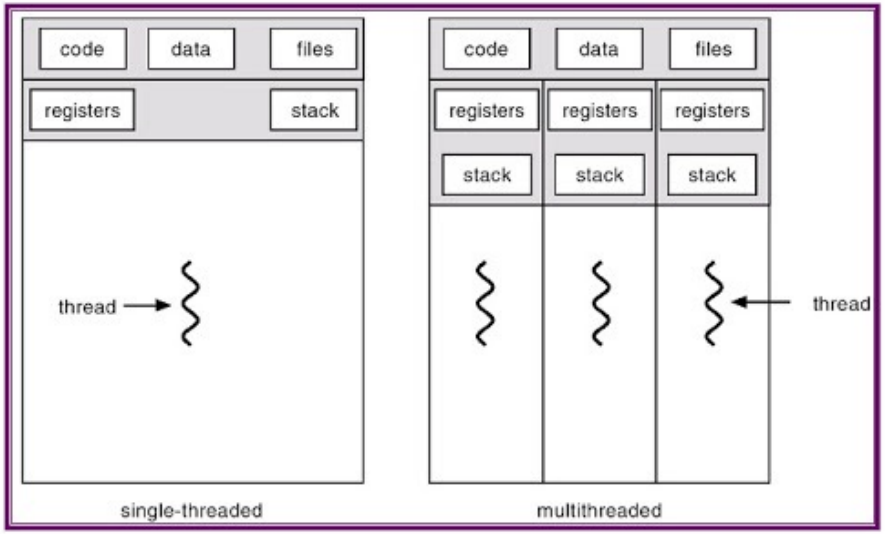
Implementações Híbridas de Thread

- Modelo M:N
  - Combina características vantajosas das duas abordagens anteriores.
- Há dois níveis de escalonamento
  - Nível do Usuário e Nível do Sistema.
  - Há M threads do usuário e N threads do sistema.
    - Geralmente M > N
- A dificuldade reside em como mapear as threads do usuário na threads do sistema.



Por que Utilizar Threads?

- Uso mais eficiente dos recursos do sistema
  - Menores *overheads* de criação, destruição e mudança de contexto do que os processos.
  - Mecanismos de comunicação mais simples.
    - Threads em geral implementam funções.
    - Threads compartilhas as variáveis globais do seu processo
    - → Por outro lado, há a necessidade de se ter mais cuidado ao acesso dessas variáveis, para manter a consistência da aplicação.



O que é uma Thread afinal?

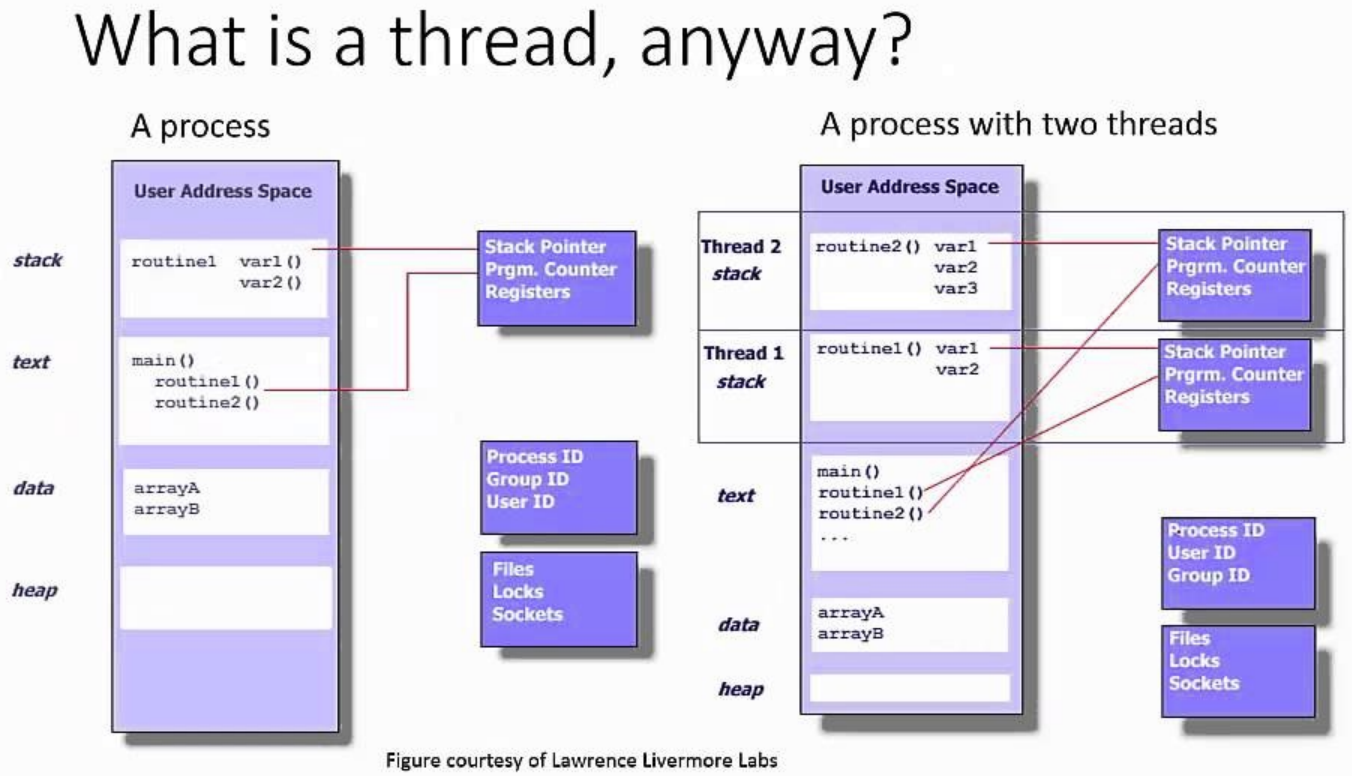
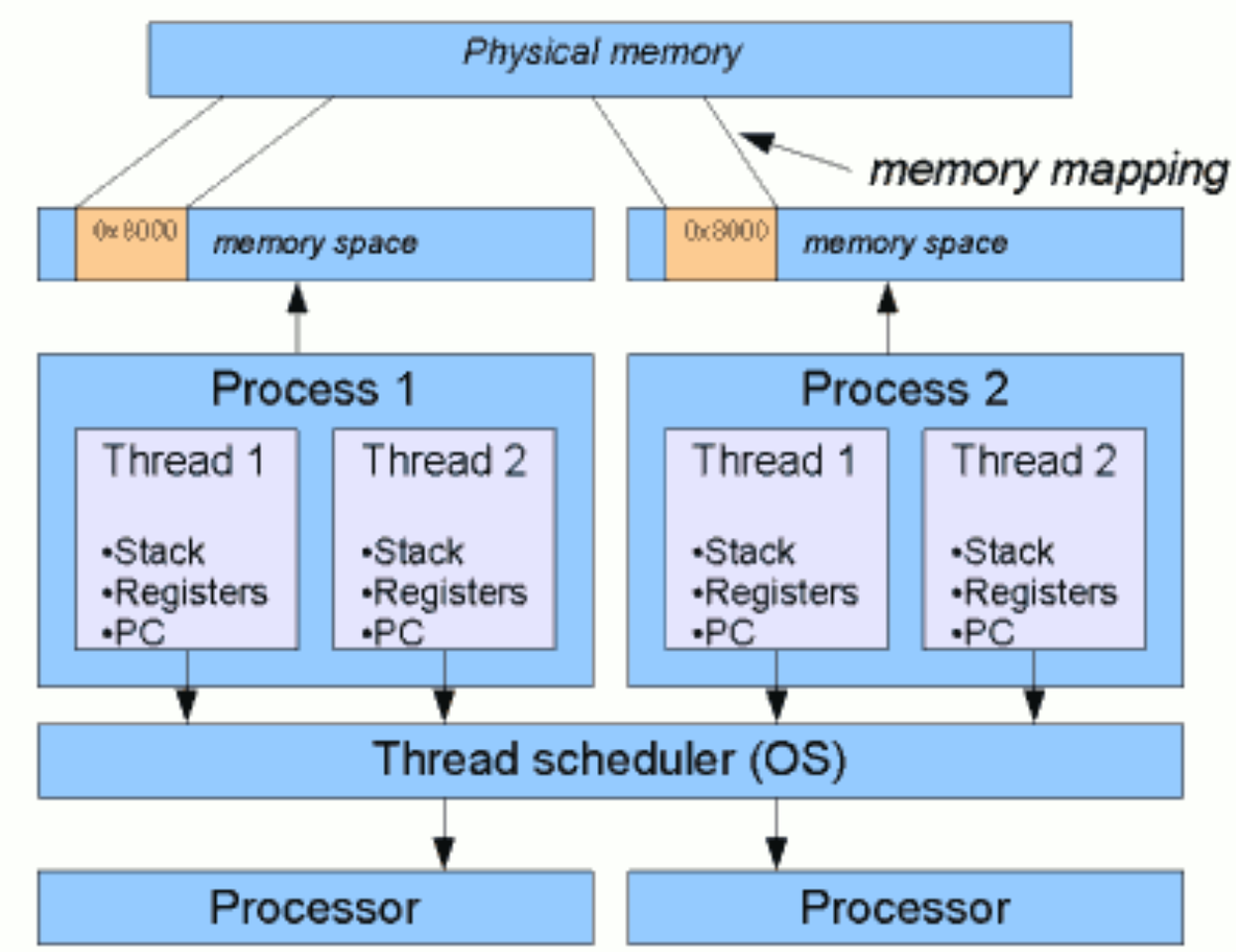


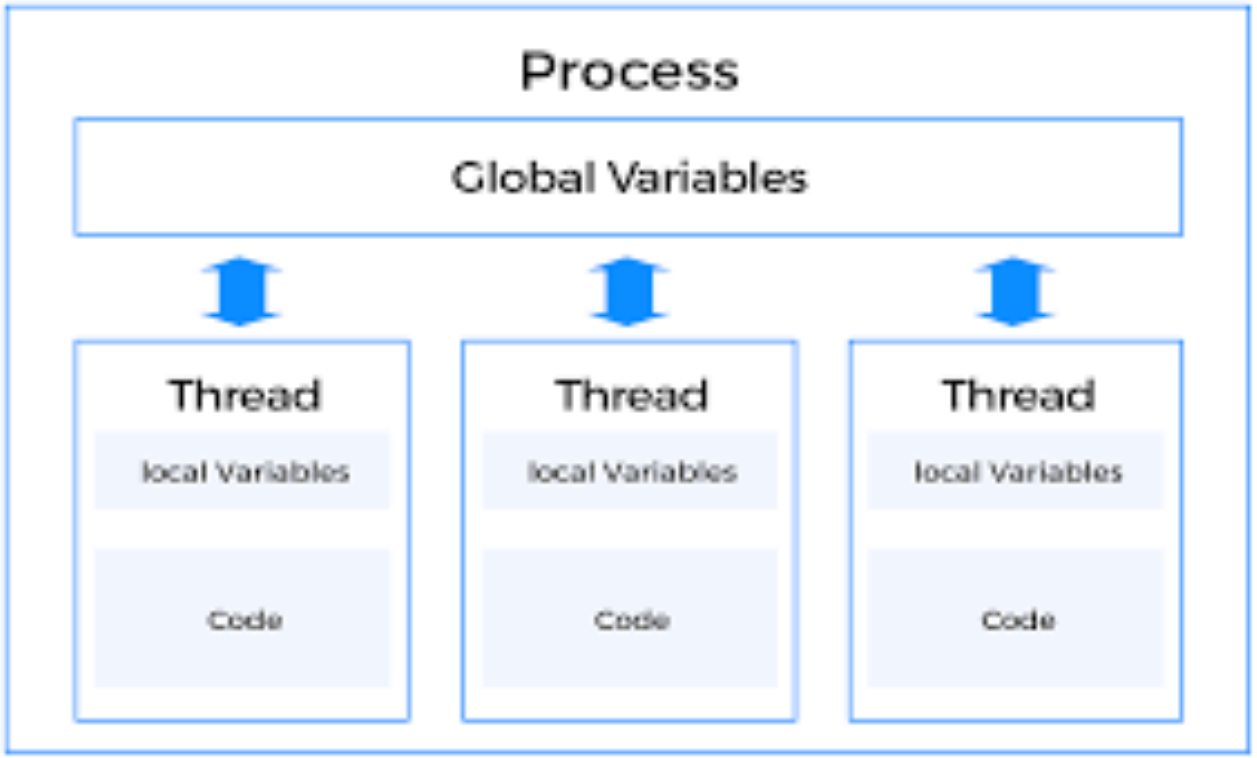
Figure courtesy of Lawrence Livermore Labs

Threads are scheduled on cores by the operating system.

O que é uma Thread afinal?



O que é uma Thread afinal?



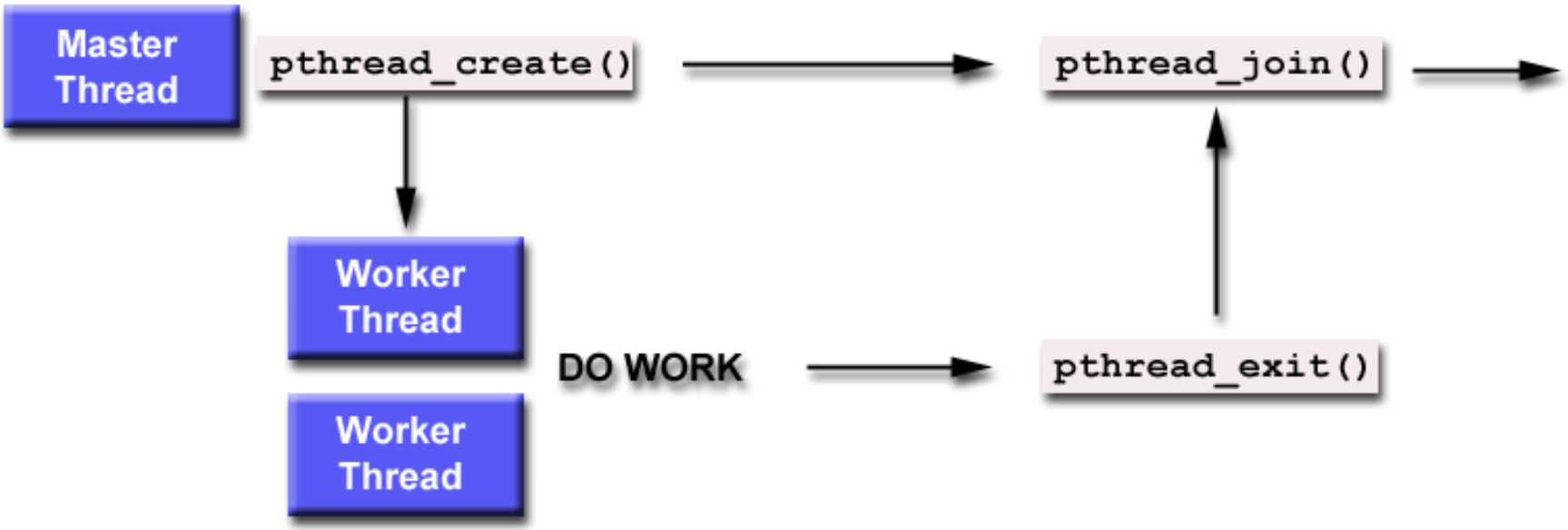
Biblioteca PTRHEAD Posix

- POSIX threads é um padrão POSIX para threads, o qual define uma API padrão para criar e manipular threads.
- As bibliotecas que implementam a POSIX threads são chamadas Pthreads, sendo muito difundidas no universo Unix e outros sistemas operacional

- https://pt.wikipedia.org/wiki/POSIX_Threads is semelhantes como Linux.

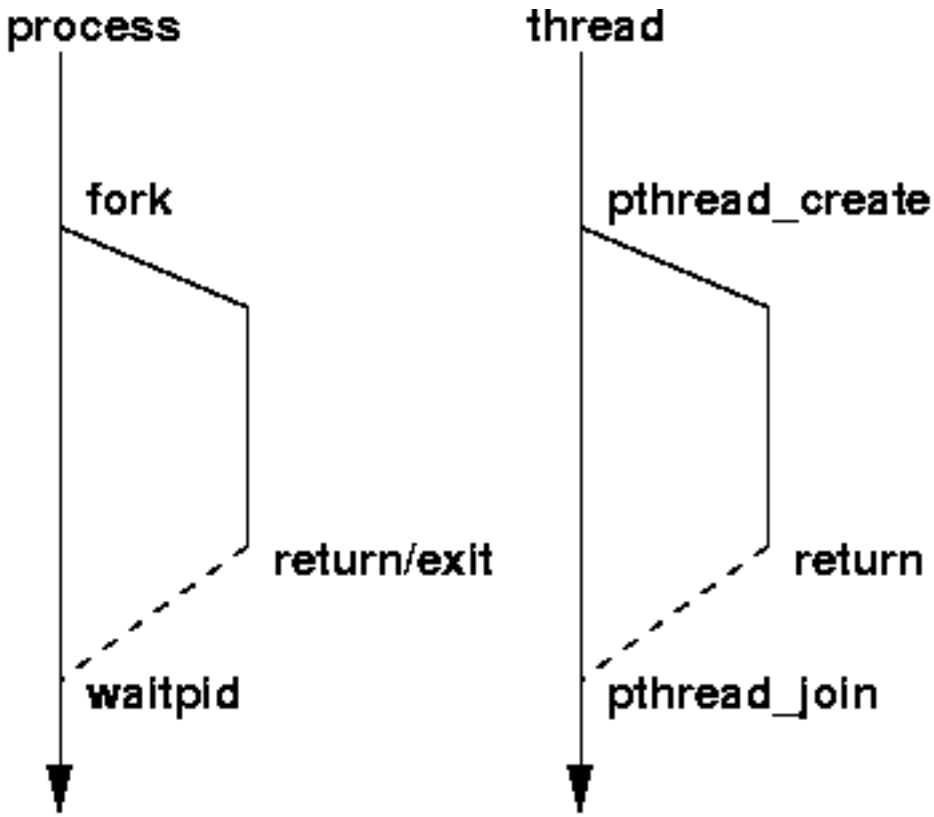
Biblioteca PTRHEAD Posix

- Uso de threads
  - Criação e término



Biblioteca PTRHEAD Posix

- Processos X Threads
  - Criação e término



Biblioteca PTRHEAD Posix

- Uso de threads
  - Execução.
    - Threads executam funções
    - Necessidade de se passar a referência da função para a Thread.

NAME top

pthread_create - create a new thread

SYNOPSIS top

```
#include <pthread.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start_routine) (void *), void *arg);

Compile and link with -pthread.
```

DESCRIPTION top

The **pthread_create**() function starts a new thread in the calling process. The new thread starts execution by invoking *start_routine*(); *arg* is passed as the sole argument of *start_routine*().

Processos x Threads

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

Biblioteca PTRHEAD Posix

- Exemplo de Criação, Término e Junção de Threads
  - Compile e execute o programa thread01.cpp
    - g++ -o thread01 thread01.cpp -lpthread
    - ./thread01
    - Observe o código e analise o resultado.
  - Descomente a parte referente a junção de threads na main(). Então compile e execute o programa thread01.cpp
    - g++ -o thread01 thread01.cpp -lpthread
    - ./thread01
    - Observe o código e analise o resultado.

Biblioteca PTRHEAD Posix

- Exemplo de Criação, Término e Junção de Threads
  - Compile e execute o programa thread02.cpp
    - g++ -o thread02 thread02.cpp -lpthread
    - ./thread02
    - Observe o código e analise o resultado.
    - Utilize o programa htop para melhor analisar o comportamento do programa.
    - Uso de variável global.

Biblioteca PTRHEAD Posix

- Exemplo de Criação, Término e Junção de Threads
  - Compile e execute o programa thread03.cpp
    - g++ -o thread03 thread03.cpp -lpthread
    - ./thread03
    - Observe o código e analise o resultado.
    - Utilize o programa htop para melhor analisar o comportamento do programa.

    - Uso de variável passada como parâmetro.

Biblioteca PTRHEAD Posix

- Exemplo de Criação, Término e Junção de Threads
  - Compile e execute o programa thread04.cpp
    - g++ -o thread04 thread04.cpp -lpthread
    - ./thread04
    - Observe o código e analise o resultado.
    - Utilize o programa htop para melhor analisar o comportamento do programa.

    - Criação de um vetor de threads

Biblioteca PTRHEAD Posix

- Exemplo de Criação, Término e Junção de Threads
  - Compile e execute o programa thread05.cpp
    - g++ -o thread05 thread05.cpp -lpthread
    - ./thread05
    - Observe o código e analise o resultado.
    - Utilize o programa htop para melhor analisar o comportamento do programa.

    - Cancelamento de threads

Biblioteca PTRHEAD Posix

- Exemplo de configuração de prioridade de Threads
  - Compile e execute o programa thread06.cpp
    - g++ -o thread06 thread06.cpp -lpthread
    - ./thread06
    - Observe o código e analise o resultado.
    - Utilize o programa htop para melhor analisar o comportamento do programa.

    - Prioridade de threads

PTRHEAD Posix – Política de Escalonamento

- Múltiplas filas de prioridade
- Quando uma thread é execução é preemptada, ela é inserido no início da sua fila.
- Quando uma thread bloqueada passa a apta, ela é inserida no final da sua fila.
- Quando uma thread troca de prioridade, ela é inserida no final de sua nova fila de prioridade.
- Quando uma thread em execução libera espontaneamente a CPU para outra thread, ela é inserida no final da sua fila.

Class Thread – Disponível a partir da versão 11 do C++

- Fornece uma API de alto nível para se manipular threads.
  - É implementada sobre a pthread Posix.
  - #include <thread>
  - Possui todas as funcionalidades da Pthread Posix.
    - Criação, cancelamento, junção, transferência, etc.
  - Procedimento de uso:
    - Criar thread (tipo de dados da #include <thread>).
    - Passar uma função, com ou sem parâmetros, paro a thread.
    - Ou passar um objeto (executará um método específico do objeto.

Class Thread – Disponível a partir da versão 11 do C++

- Exemplo de criação, uso e junção (join) de uma thread da Class thread
  - Compile e execute o programa tClass01.cpp
    - g++ --std=c++11 -o tClass01 tClass01.cpp -lpthread
    - ./tClass01
    - Observe o código e analise o resultado.
    - Utilize o programa htop para melhor analisar o comportamento do programa.
    - Threads têm PID?

Class Thread – Disponível a partir da versão 11 do C++

- Exemplo de criação, uso e junção (join) de duas threads da Class thread. A função de uma delas possui parâmetro.
  - Compile e execute o programa tClass02.cpp
    - g++ --std=c++11 -o tClass02 tClass02.cpp -lpthread
    - ./tClass02
    - Observe o código e analise o resultado.
    - Utilize o programa htop para melhor analisar o comportamento do programa.

Class Thread – Disponível a partir da versão 11 do C++

- Exemplo demonstrando diversas formas de se criar threads da Class thread.
  - Compile e execute o programa tClass03.cpp
    - g++ --std=c++11 -o tClass03 tClass03.cpp -lpthread
    - ./tClass03
    - Observe o código e analise o resultado.
    - Utilize o programa htop para melhor analisar o comportamento do programa.

Class Thread – Disponível a partir da versão 11 do C++

- Exemplo de como mudar prioridade em threads da Class thread.
  - Compile e execute o programa tClass04.cpp
    - g++ --std=c++11 -o tClass04 tClass04.cpp -lpthread
    - ./tClass04
    - Observe o código e analise o resultado.
    - Utilize o programa htop para melhor analisar o comportamento do programa.

Class Thread – Disponível a partir da versão 11 do C++

- Exercício: Utilize o programa anterior (tClass04.cpp) como base para investigar como funciona o mecanismo de escalonamento das threads.
  - Gere duas threads.
  - Faça como que as duas threads executem na mesma CPU.
  - Atribua prioridades diferentes para as duas threads.
  - Observe seus comportamentos (para isto faça que elas escrevam seus Ids, por exemplo).
  - Observe o comportamento pelo programa htop.