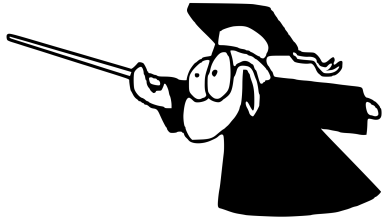
Gerência de Processos - Threads

EMB5632 - Sistemas Operacionais
Prof. Dr. Ricardo José Pfitscher
ricardo.pfitscher@ufsc.br

Objetivos de aprendizagem



- Entender o que é uma thread
- Entender a diferença entre processo e thread
- Codificar um programa multithread

UFSC

Cronograma

- Conceito
- Multiprogramação
- Criação de processos /Término de Processos
- Hierarquias de Processos
- Estados de Processos
- Implementação de Processos

UFSC

Cronograma

- **Conceito Threads**
- Compartilhamento de recursos
- Vantagens do uso de threads
- Problemas do uso de threads
- Exemplos de uso de threads
- Implementação de threads
- Convertendo um código Monothread em Multithread

Conceito [1/3]

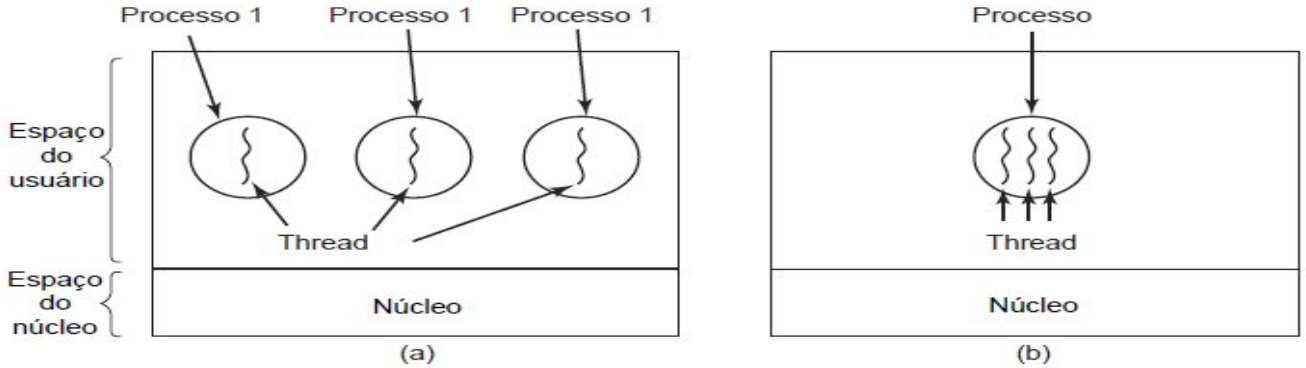
- O modelo de thread [1/3]
  - Processos:
    - Cada processo possui:
      - Espaço de endereçamento
      - Um contador de programa e um fluxo de controle (thread)
    - Processos agrupam recursos
      - Espaço de endereçamento (código+dados), arquivos, processos filhos...
      - Esse agrupamento facilita o gerenciamento
  - Thread:
    - Representa o estado atual de execução de um processo
      - Contador de programa, registradores, pilha
    - Podemos dizer que é a parte “executante” do processo
      - Diversas linhas de controle

Conceito [2/3]

- O modelo de thread [2/3]
  - O uso de múltiplas threads
    - Execução paralela sobre o mesmo recurso
    - Idéia semelhante aos processos em paralelo
  - Threads também são chamadas de processos leves
    - Possuem algumas propriedades dos processos
  - Multithread
    - Existência de múltiplas threads em um mesmo processo

Conceito [3/3]

- O modelo de thread [3/3]
  - Ex.: Três threads



(a) Processos tradicionais:

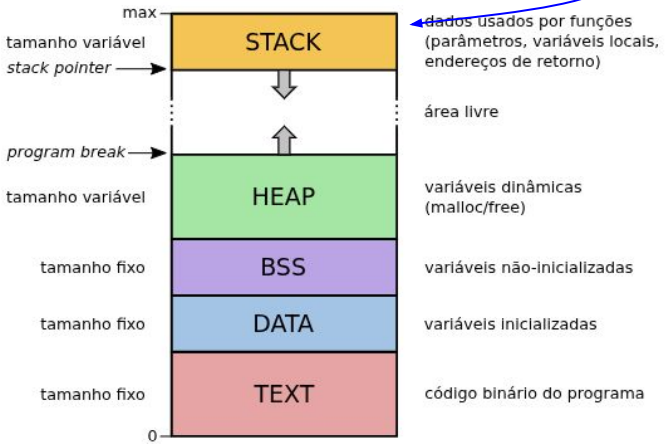
- Cada um com seu próprio espaço de endereçamento
- Uma thread de controle

(b) Processo Multithread:

- O processo possui um espaço de endereçamento, compartilhado entre as threads
- Três threads de controle

Conceito [3/3]

- O modelo de thread



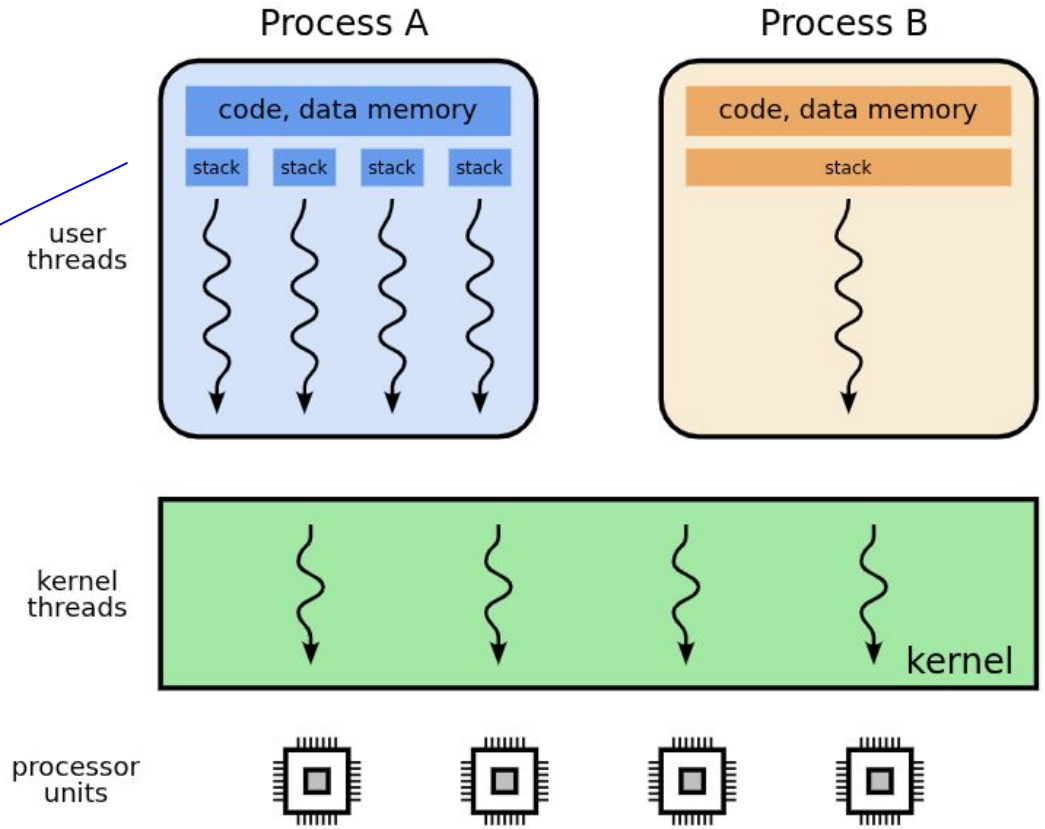


Figura 5.4: *Threads* de usuário e de núcleo.

Fonte: (Maziero, 2019)

Cronograma

- ~~**Conceito Threads**~~
- **Compartilhamento de recursos**
- Vantagens do uso de threads
- Problemas do uso de threads
- Exemplos de uso de threads
- Implementação de threads
- Convertendo um código Monothread em Multithread

Compartilhamento de recursos [1/3]

- As threads de um processo compartilham muitos dos recursos do processo
  - Compartilham o mesmo **espaço de endereçamento**
  - Não existe proteção entre threads
    - Um thread pode:
      - Ler, escrever, apagar valores da pilha de outro thread
      - Ex.: Se um thread abre um arquivo, os outros threads podem ler e escrever nele.
    - A unidade de gerenciamento de recursos é o processo.
      - Se cada thread tivesse seu próprio espaço de endereçamento, alarmes, arquivos abertos, etc. **Ele seria um processo**
    - A base das threads: Compartilhamento de recursos de um processo entre múltiplos fluxos de execução

Compartilhamento de recursos [2/3]

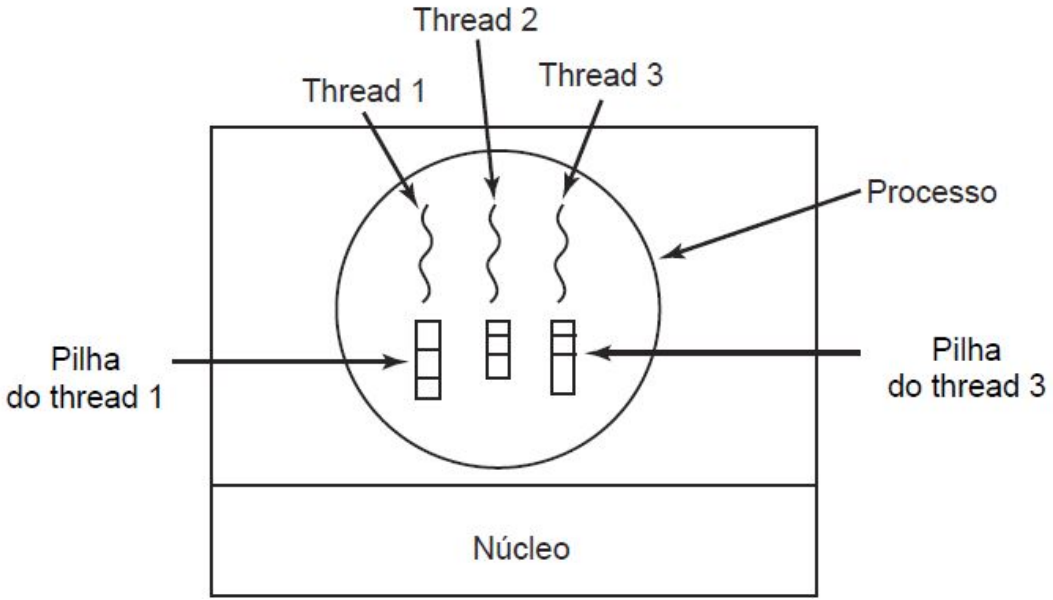
Itens por processo	Itens por thread
Espaço de endereçamento	Contador de programa
Variáveis globais	Registradores
Arquivos abertos	Pilha
Processos filhos	Estado
Alarmes pendentes	
Sinais e tratadores de sinais	
Informação de contabilidade	

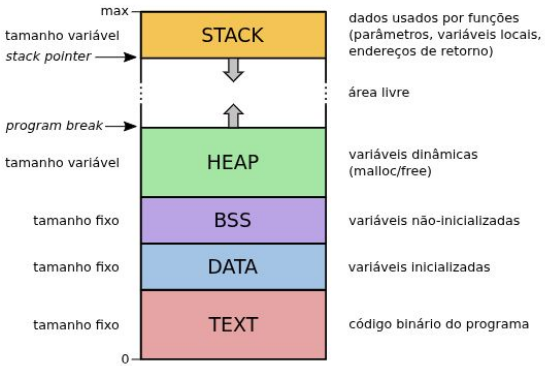
Itens compartilhados por todas as threads no processo

Itens específicos a cada thread

Compartilhamento de recursos [3/3]

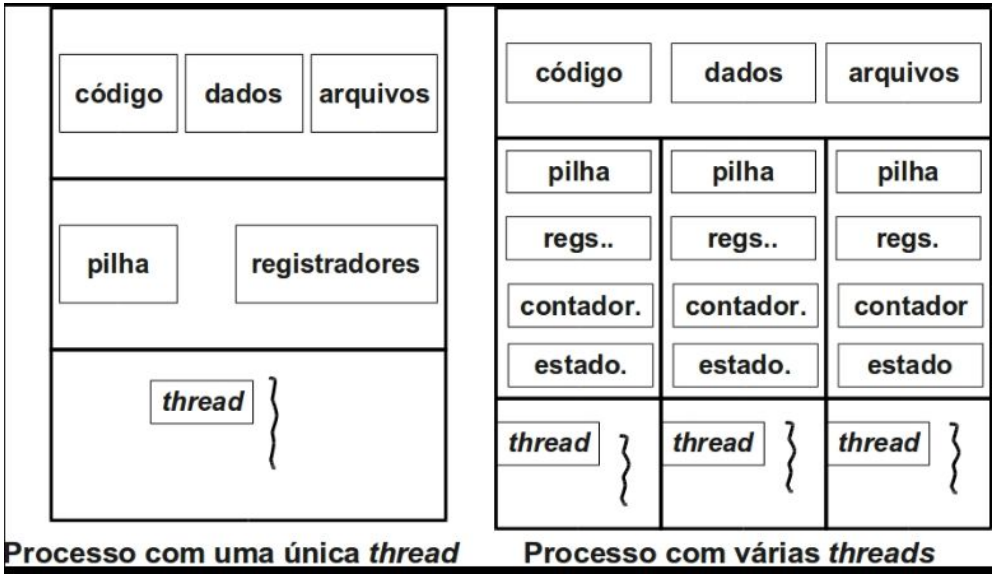
- Cada thread precisa da sua própria pilha
  - Uma estrutura para cada rotina chamada, que ainda não retornou
  - Mantém suas variáveis locais e seu histórico de execução

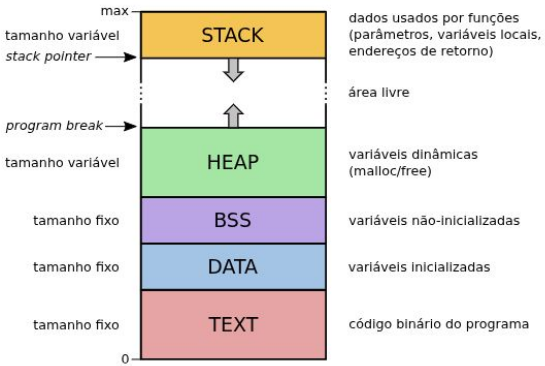




Compartilhamento de recursos [3/3]

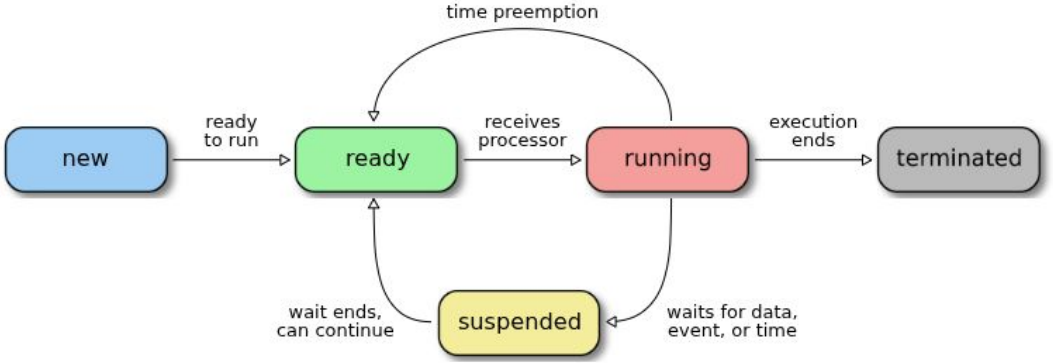
- Cada thread precisa da sua própria pilha
  - Uma estrutura para cada rotina chamada, que ainda não retornou
  - Mantém suas variáveis locais e seu histórico de execução





Estados de um thread

- Assim como processos threads tem estados:
  - Em execução, pronto e bloqueado



- Dependendo da implementação, bloquear uma thread de um processo, irá bloquear as demais

Cronograma

- ~~Conceito Threads~~
- ~~Compartilhamento de recursos~~
- **Vantagens do uso de threads**
- Problemas do uso de threads
- Exemplos de uso de threads
- Implementação de threads
- Convertendo um código Monothread em Multithread

Vantagens do uso de Threads

- Possibilidade de soluções paralelas para problemas
  - Cada thread se preocupa com uma parte do problema
  - Melhor em aplicações dirigidas a eventos
- Desempenho
  - Criar e destruir threads é mais rápido que processos
  - O chaveamento de contexto é muito mais rápido
  - Permite combinar threads I/O-bound e CPU-bound
  - É mais fácil compartilhar variáveis com Threads que com processos

e mais perigoso

Cronograma

- ~~Conceito Threads~~
- ~~Compartilhamento de recursos~~
- ~~Vantagens do uso de threads~~
- **Problemas do uso de threads**
- Exemplos de uso de threads
- Implementação de threads
- Convertendo um código Monothread em Multithread

UFSC

Problemas com threads

- Complicações no Modelo de Programação
  - Um processo filho herda todas as threads de um processo pai?
  - Se herdar, o que acontece quando a thread bloqueia por uma entrada de teclado?
- Complicações pelos recursos compartilhados
  - Se uma thread fecha um arquivo que está sendo usado por outra?
  - Se uma thread começa uma alocação de memória e acaba sendo substituída por outra?
  - Se um thread modifica um valor de um vetor que está sendo lido por outra?

Cronograma

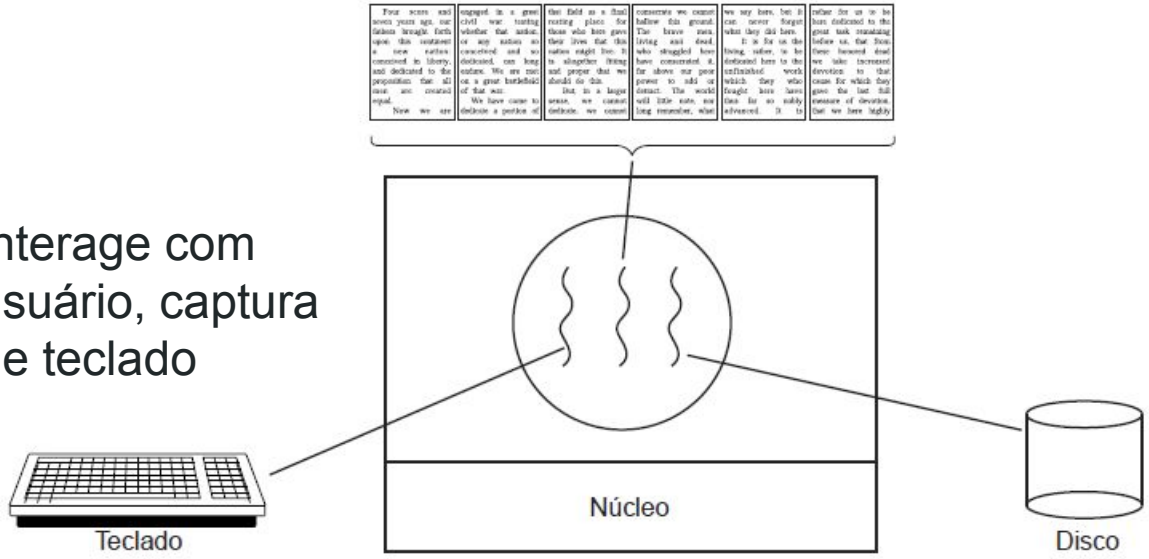
- ~~Conceito Threads~~
- ~~Compartilhamento de recursos~~
- ~~Vantagens do uso de threads~~
- ~~Problemas do uso de threads~~
- **Exemplos de uso de threads**
- Implementação de threads
- Convertendo um código Monothread em Multithread

Exemplos de uso de Threads [1/4]

- Processador de texto [1/2]
  - Considere a implementação monothread
  - Um usuário precisa modificar uma sentença da primeira página de um livro de 800 páginas.
  - Após verificar se a formatação ficou correta na página, ele decide fazer uma mudança na página 600
    - Para chegar lá, o processador de textos terá de formatar todo o conteúdo até ela.
      - Ele não sabe qual será a primeira linha da página 600 se não o fizer
  - Demora substancial
    - Irritação do usuário

Exemplos de uso de Threads [2/4]

- Processador de texto [2/2]
  - Agora com três threads



Faz a formatação em segundo plano

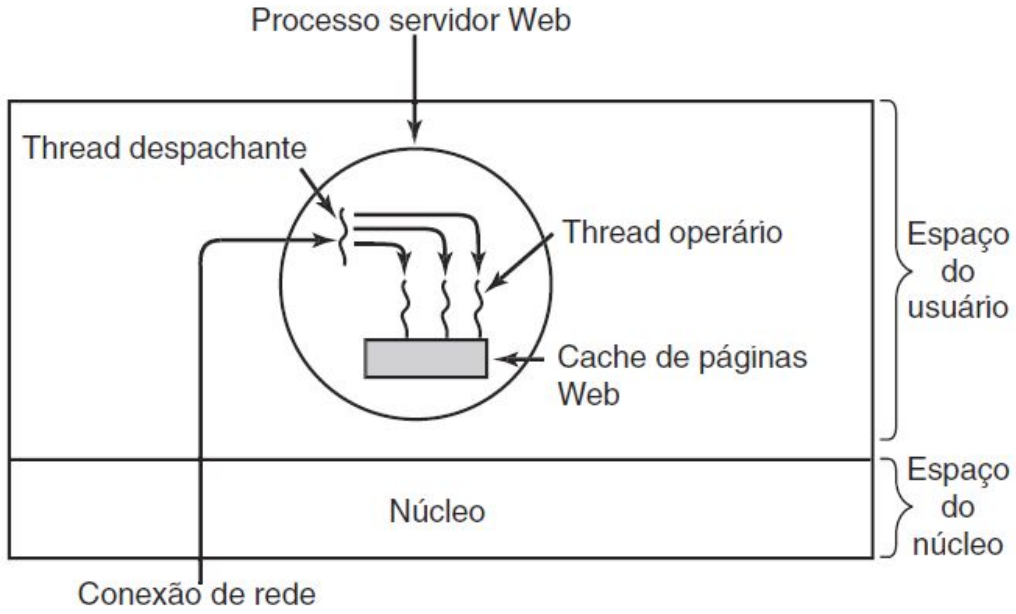
Interage com usuário, captura de teclado

Salva os arquivos no disco (backup)

Exemplos de uso de Threads [3/4]

- Servidor *web* multithreaded [1/2]

Executa em um laço infinito, recebendo requisições de trabalho e entregando a um operário



O operário verifica se a página está disponível em cache, se sim retorna ao cliente e bloqueia, se não, busca no disco, retorna e bloqueia aguardando nova requisição

Exemplos de uso de Threads [4/4]

- Servidor *web* multithreaded [2/2]
  - Código simplificado

```
while (TRUE) {
    get_next_request(&buf);
    handoff_work(&buf);
}
```

(a) despachante

```
while (TRUE) {
    wait_for_work(&buf)
    look_for_page_in_cache(&buf, &page);
    if(page_not_in_cache(&page))
        read_page_from_disk(&buf, &page);
    return_page(&page);
}
```

(b) operário

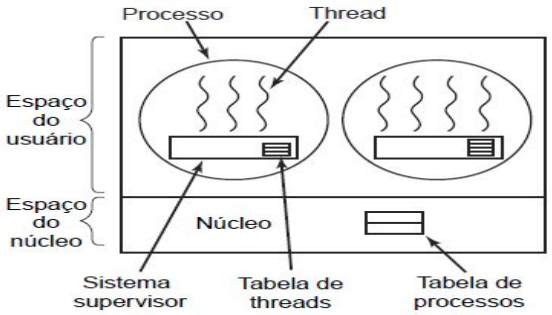
Cronograma

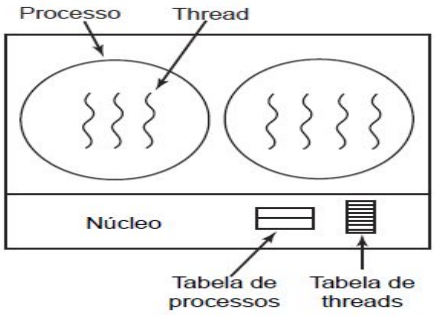
- ~~Conceito Threads~~
- ~~Compartilhamento de recursos~~
- ~~Vantagens do uso de threads~~
- ~~Problemas do uso de threads~~
- ~~Exemplos de uso de threads~~
- **Implementação de threads**
- Convertendo um código Monothread em Multithread

UFSC

Implementação de threads

- Duas formas de se implementar
  - No espaço do usuário (N:1)
    - Gerenciado ao nível de usuário
  - No espaço do núcleo (1:1)
    - Gerenciado ao nível de SO (conhece as threads)
  - Modelo híbrido (N:M)
    -





Threads no espaço do usuário

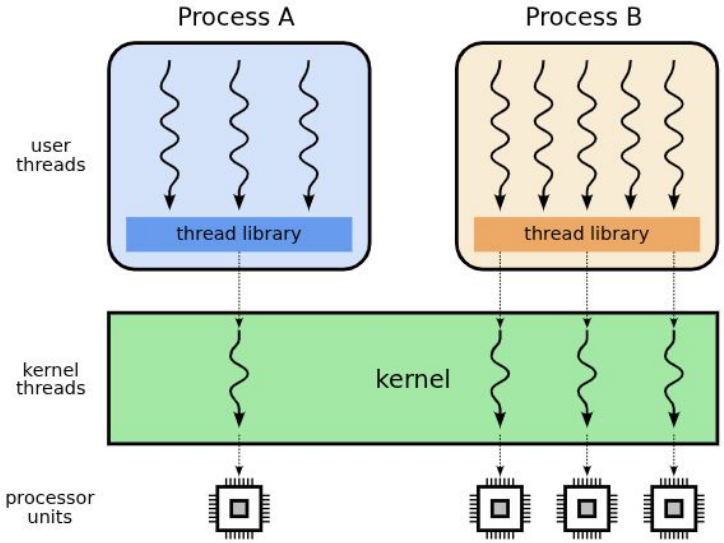
- São implementadas por uma biblioteca, o núcleo não sabe nada sobre elas
  - **N** threads são mapeadas em um mesmo processo (N:1)
  - Núcleo escalona processos, não threads
  - O escalonamento das threads é feito pela biblioteca
  - Cada processo tem uma **Tabela de threads**
- Vantagens
  - Permite usar threads em SOs que não tem suporte
  - Chaveamento de contexto entre threads não requer chamadas de sistema Desempenho
- Desvantagens
  - Tratamento de chamadas bloqueantes
    - Uma chamada de sistema de uma thread para bloqueio, a fim de fazer uma leitura de teclado, isso irá bloquear todas as outras threads do processo...
  - Preempção por tempo é complicada
    - Se uma thread de um processo começar a executar, as outras só irão executar se a primeira abdicar voluntariamente
  - Em ambos os casos são necessárias modificações, ou no SO, ou na biblioteca para tratar os problemas

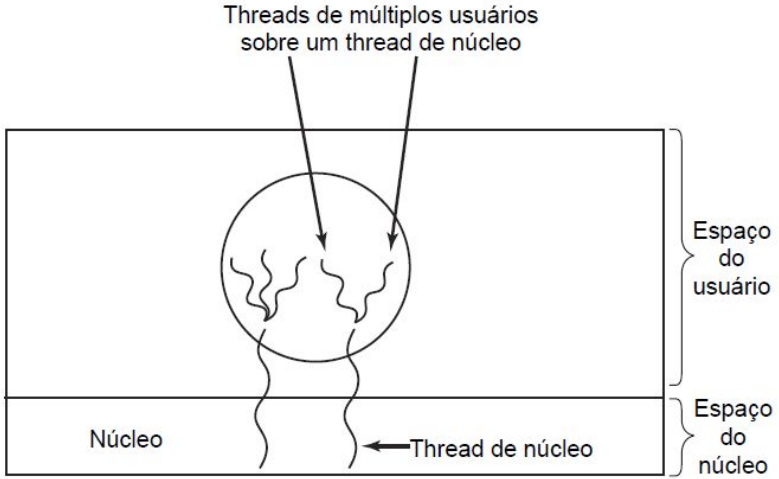
Threads no espaço do núcleo

- O núcleo conhece e escalona as threads
  - Não há necessidade de biblioteca
- Vantagens
  - Facilidade para lidar com chamadas bloqueantes
  - Preempção entre threads
- Desvantagens:
  - Desempenho, as operações que envolvem threads têm um custo maior
    - Exigem chamadas ao núcleo
    - Ex.: Criar novas threads

Threads híbridas

- Combina ambos os modelos anteriores
  - O núcleo conhece somente as suas threads
    - Estas são multiplexadas para o usuário
    - Cada thread de usuário aguarda sua vez para usar a thread de núcleo





Modelos de threads

Modelo	N:1	1:1	N:M
Implementação	biblioteca	núcleo	ambos
Complexidade	baixa	média	alta
Custo de gerência	nulo	médio	alto
Escalabilidade	alta	baixa	alta
Vários processadores	não	sim	sim
Trocas de contexto	rápida	lenta	ambos
Divisão de recursos	injusta	justa	variável
Exemplos	GNU Threads	Windows Linux	Solaris FreeBSD

Fonte: (Maziero, 2019)

Cronograma

- ~~Conceito Threads~~
- ~~Compartilhamento de recursos~~
- ~~Vantagens do uso de threads~~
- ~~Problemas do uso de threads~~
- ~~Exemplos de uso de threads~~
- ~~Implementação de threads~~
- Convertendo um código Monothread em Multithread

Convertendo código para Multithreading

- Problemas em potencial
  - Variáveis globais modificadas por várias threads
    - Proibir o uso de variáveis globais
    - Permitir variáveis globais privativas de cada thread
  - Bibliotecas não reentrantes: funções que não podem ser executadas por mais de uma thread
    - Permitir apenas uma execução por vez
  - Sinais:
    - Quem captura? Como tratar?
  - Gerenciamento da pilha
    - O sistema precisa tratar o overflow de várias pilhas

Leitura Cap. 2.2.9 Tanenbaum

Threads em C

```c
// Exemplo de uso de threads Posix em C no Linux
// Compilar com gcc exemplo.c -o exemplo -lpthread

#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define NUM_THREADS 5

// cada thread vai executar esta função
void *print_hello (void *threadid)
{
   printf ("%ld: Hello World!\n", (long) threadid);
   sleep (5) ;
   printf ("%ld: Bye bye World!\n", (long) threadid);
   pthread_exit (NULL);     // encerra esta thread
}
```

```c
// thread "main" (vai criar as demais threads)
int main (int argc, char *argv[])
{
   pthread_t thread[NUM_THREADS];
   long status, i;

   // cria as demais threads
   for(i = 0; i < NUM_THREADS; i++)
   {
      printf ("Creating thread %ld\n", i);
      status = pthread_create (&thread[i], NULL, print_hello, (void *) i);

      if (status) // ocorreu um erro
      {
         perror ("pthread_create");
         exit (-1);
      }
   }

   // encerra a thread "main"
   pthread_exit (NULL);
}
```

Threads em Java

```
1  public class MyThread extends Thread {
2     int threadID;
3     private static final int NUM_THREADS = 5 ;
4
5     MyThread (int ID) {
6        threadID = ID;
7     }
8
9     public void run () {                                  // corpo de cada thread
10       System.out.println (threadID + ": Hello World!") ;
11       try {
12         Thread.sleep(5000);
13       }
14       catch (InterruptedException e) {
15         e.printStackTrace();
16       }
17       System.out.println (threadID + ": Bye bye World!") ;
18    }
19
20    public static void main (String args[]) {
21       MyThread[] t = new MyThread[NUM_THREADS] ;
22
23       for (int i = 0; i < NUM_THREADS; i++) {           // cria as threads
24          t[i] = new MyThread (i);
25       }
26       for (int i = 0; i < NUM_THREADS; i++) {           // inicia a execução das threads
27          t[i].start () ;
28       }
29    }
30 }
```

Fonte: (Maziero, 2019)

E aí, threads ou processos?

Característica	Com processos	Com *threads* (1:1)	Híbrido
Custo de criação de tarefas	alto	baixo	médio
Troca de contexto	lenta	rápida	variável
Uso de memória	alto	baixo	médio
Compartilhamento de dados entre tarefas	canais de comunicação e áreas de memoria compartilhada.	variáveis globais e dinâmicas.	ambos.
Robustez	um erro fica contido no processo.	um erro pode afetar todas as *threads*.	um erro pode afetar as *threads* no mesmo processo.
Segurança	cada processo pode executar com usuários e permissões distintas.	todas as *threads* herdam as permissões do processo onde executam.	*threads* com as mesmas permissões podem ser agrupadas em um mesmo processo.
Exemplos	Apache 1.*, PostGres	Apache 2.*, MySQL	Chrome, Firefox, Oracle

UFSC

Exercício (Moodle)

- Pesquise sobre a função pthread_join e explique o que ela faz;
- Faça um programa que compute a soma dos valores em um vetor de tamanho N utilizando threads
    - Divida a tarefa de calcular pelo número de threads que você criar → lembre de não acessar a mesma área do vetor
    - Meça através do comando time do terminal o tempo de execução do seu programa
        - Avalie se há ganho de desempenho por usar mais de uma thread comparado a uma thread única

Referências

- *MAZIERO, C. Sistemas Operacionais: Conceitos e Mecanismos. Editora da UFPR, 2019. 456 p. ISBN 978-85-7335-340-2*
- Andrew S. Tanenbaum. Sistemas Operacionais Modernos, 3a Edição. Capítulo 2.Pearson Prentice-Hall, 2009.

UFSC