INSTITUTO FEDERAL
RIO GRANDE DO SUL

Sistemas Operacionais

- Tarefas
- Processos
- Threads

Apresentações Adaptadas do Material do Prof. Marcelo Paravisi

Tarefas x Processos

- Agenda da aula de hoje
  - Tarefas
  - Processos
  - Threads

Bibliografia Base:
Livro Eletronico Maziero, Carlos Alberto Sistemas operacionais:
conceitos e mecanismos [recurso eletrônico] / Carlos Alberto Maziero.
– Curitiba : DINF - UFPR, 2019.

INSTITUTO FEDERAL
RIO GRANDE DO SUL

PROBLEMA

Em um sistema de computação, é frequente a necessidade de executar várias tarefas distintas simultaneamente.

Por exemplo:

• O usuário de um computador pessoal pode estar editando uma imagem, imprimindo um relatório, ouvindo música e trazendo da Internet um novo software, tudo ao mesmo tempo.

• Em um grande servidor de e-mails, milhares de usuários conectados remotamente enviam e recebem e-mails através da rede.

• Um navegador Web precisa buscar os elementos da página a exibir, analisar e renderizar o código HTML e os gráficos recebidos, animar os elementos da interface e responder aos comandos do usuário.

INSTITUTO FEDERAL
RIO GRANDE DO SUL

PROBLEMA

Um sistema de computação quase sempre tem mais atividades a executar que o número de processadores disponíveis!

SOLUÇÃO

Cada tarefa receba uma quantidade de processamento que atenda suas necessidades.

INSTITUTO FEDERAL
RIO GRANDE DO SUL

TAREFA

Uma tarefa é definida como sendo a execução de um fluxo sequencial de instruções, construído para atender uma finalidade específica.

PROGRAMA ≠ TAREFA

INSTITUTO FEDERAL
RIO GRANDE DO SUL

TAREFA

- **Um programa** é um conjunto de uma ou mais sequências de instruções escritas para resolver um problema específico, constituindo assim uma aplicação ou utilitário.

- **Uma tarefa** é a execução, pelo processador, das sequências de instruções definidas em um programa para realizar seu objetivo.

TIPOS: THREADS, PROCESSOS

INSTITUTO FEDERAL
RIO GRANDE DO SUL

TAREFA

- **O PROGRAMA**: representa um **conceito estático**, sem um estado interno definido (que represente uma situação específica da execução) e sem interações com outras entidades (o usuário ou outros programas)

- **A TAREFA:** Trata-se de um conceito dinâmico, que **possui um estado interno** bem definido a cada instante (os valores das variáveis internas e a posição atual da execução) e interage com outras entidades: o usuário, os periféricos e/ou outras tarefas.

TAREFA

Analogia com Receita de Bolo:

- **O programa** é o equivalente de uma “receita de bolo” dentro de um livro de receitas (um diretório) guardado em uma estante (um disco) na cozinha (o computador).

- **“Executar”** a receita, providenciando os ingredientes e seguindo os passos definidos na receita, é a tarefa propriamente dita.


Farinha de Semente de Uva Orgânica Uvas

Ingredientes:
- 3 ovos
- 1 xícara de chá de açúcar
- 2 xícaras de chá de Farinha de trigo
- 1 xícara de chá de Farinha de Semente de Uva Orgânica UvaSó
- xícara de chá de óleo
- 1 colher de chá de Fermento químico
- 1 pitada de sal
- 1 xícara de chá de água quente

Modo de preparo:
É só bater os ingredientes (exceto o fermento) no liquidificador, acrescentando aos poucos a Farinha de trigo e a Farinha de semente de uva. Ao Final, misture o fermento em pó, coloque em uma forma untada e enfarinhada, e leve ao forno com temperatura média por 40 minutos.

Dicas da nutri:
-> Dê preferência para os ingredientes orgânicos. Além de serem mais saudáveis e ecologicamente corretos, eles são LIVRES DE TRANSGÊNICOS
-> Prefira o açúcar orgânico, mascavo ou demerara, além de não passarem por refinamento através da utilização de substâncias químicas, eles contêm minerais como Ferro e Cálcio.
-> Se optar pelo óleo de soja, prefira marcas que não utilizem soja transgênica (a marca "Leve" possui esta opção).

• A cada momento, a cozinheira (o processador) está seguindo um passo da receita (posição da execução) e tem uma certa disposição dos ingredientes e utensílios em uso (as variáveis internas d a tarefa).

O GERENCIAMENTO DA TAREFA

- Em um computador, o processador tem de **executar todas as tarefas** submetidas pelos usuários.

- Essas tarefas geralmente têm **comportamento, duração e importância** distintas.

- Cabe ao sistema operacional organizar as tarefas para executá-las e decidir em que ordem fazê-lo.

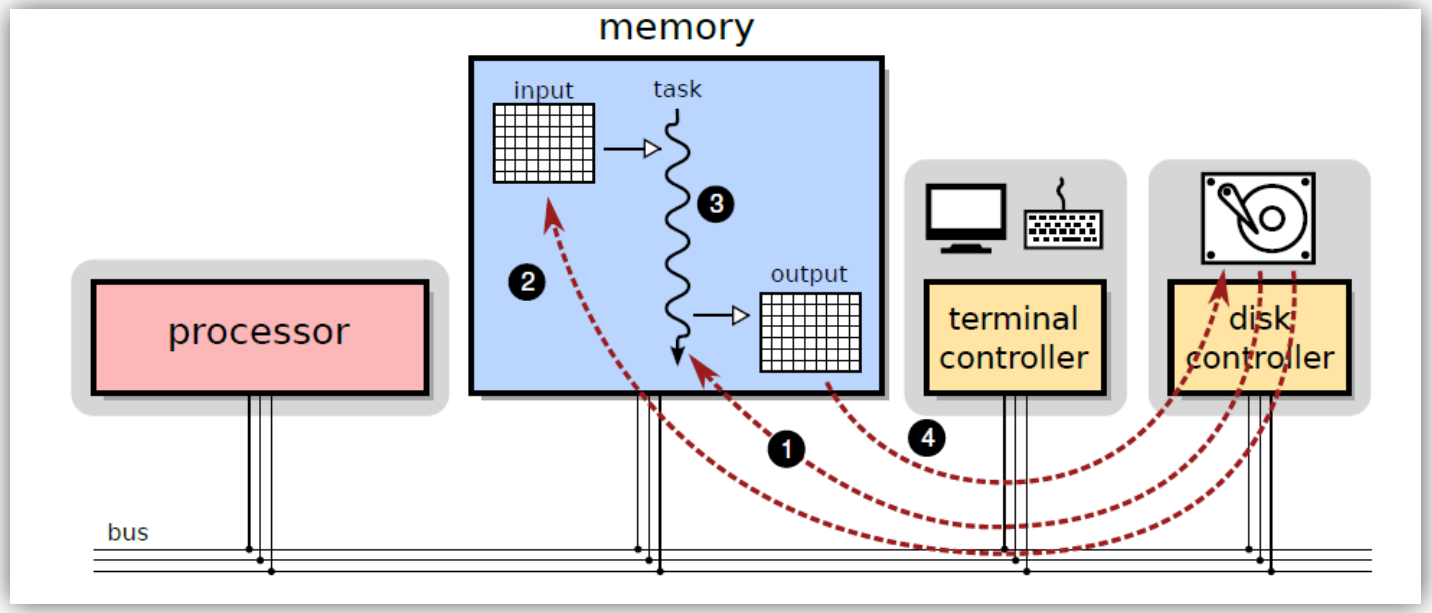
INSTITUTO FEDERAL
RIO GRANDE DO SUL

A GERENCIA DA TAREFA

- **Sistemas monotarefa**
  - Cada programa binário era carregado do disco para a memória e executado até sua conclusão.
- **Sistemas multitarefas**
  - Várias tarefas podiam estar em andamento simultaneamente: uma estava ativa (executando) e as demais prontas (esperando pelo processador) ou suspensas (esperando dados ou eventos externos).
- **Sistemas de tempo compartilhado**
  - Para cada tarefa que recebe o processador é definido um prazo de processamento, denominado fatia de tempo ou *"quantum" (uma pequena porção).*

SISTEMAS MONOTAREFA

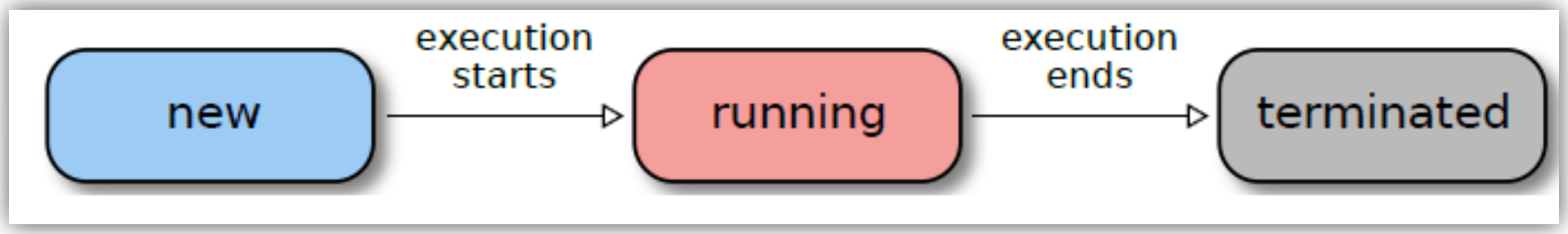
- Primeiros sistemas de computação (1940)
- 1 tarefa por vez
  - Sem monitor (Controle Humano)
  - Com monitor (FILA DE PROGRAMAS)



SISTEMA MONOTAREFA

ESTADOS DAS TAREFAS

Nesse método de processamento de tarefas é possível delinear um diagrama de estados para cada tarefa executada pelo sistema:



SISTEMA MONOTAREFA

Cada Tarefa era carregado pelo Operador.

O MONITOR DE SISTEMA

Com a evolução do hardware, as tarefas de carga e descarga de código entre memória e disco, coordenadas por um operador humano, passaram a se tornar críticas.

Mais tempo era perdido nesses procedimentos manuais que no processamento da tarefa em si.

Para resolver esse problema foi construído um **programa monitor**, que era carregado na memória no início da operação do sistema, com a função de gerenciar a execução dos demais programas.

INSTITUTO FEDERAL
RIO GRANDE DO SUL

O MONITOR DE SISTEMA

O programa monitor executava **continuamente** os seguintes passos sobre uma fila de programas a executar, armazenada no disco:

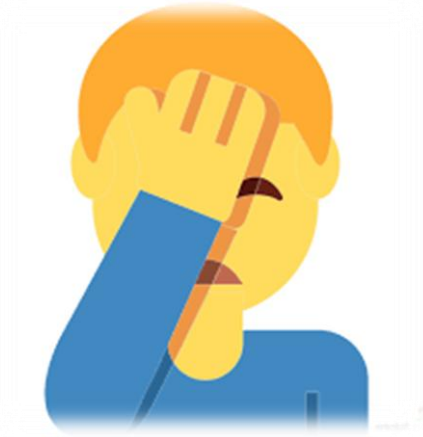
1. Carregar um programa do disco para a memória;
2. Carregar os dados de entrada do disco para a memória;
3. Transferir a execução para o programa recém carregado;
4. Aguardar o término da execução do programa;
5. Escrever os resultados gerados pelo programa no disco.

O MONITOR DE SISTEMA

Monitor agilizou o uso do processador.

Novo Problema:

Uma tarefa ficava aguardando recursos.



SISTEMAS MULTI-TAREFAS

Novo Problema:

- CPU muito mais rápida que dispositivos de E/S
- CPU ficava esperando a E/S

Solução:

PREEMPÇÃO

Processador suspende a execução da tarefa que espera dados externos **e passa a executar outra tarefa.**

INSTITUTO FEDERAL
RIO GRANDE DO SUL

SISTEMAS MULTI-TAREFAS

- A solução encontrada para resolver esse problema foi permitir ao monitor suspender a execução da tarefa que espera dados externos e passar a executar outra tarefa.

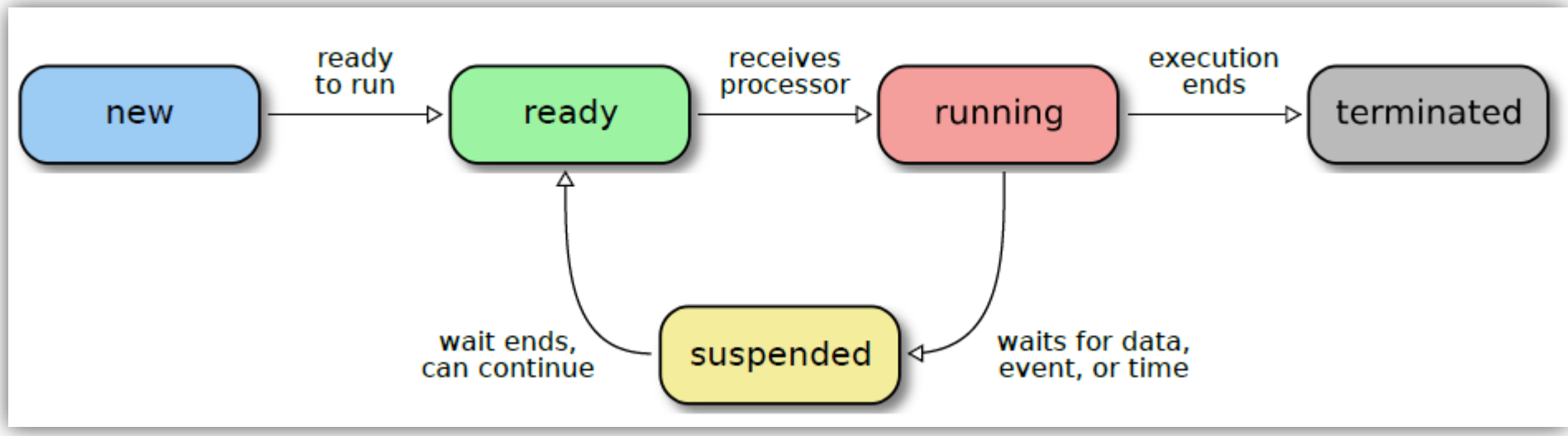
- Mais tarde, quando os dados de que a tarefa suspensa necessita estiverem disponíveis, ela pode ser retomada no ponto onde parou. Para tal, é necessário ter mais memória (para poder carregar mais de um programa ao mesmo tempo) e criar mecanismos no monitor para suspender uma tarefa e retomá-la mais tarde.

- Uma forma simples de implementar a suspensão e retomada de tarefas de forma transparente consiste no monitor fornecer um conjunto de rotinas padronizadas de entrada/saída à tarefas; essas rotinas implementadas pelo monitor recebem as solicitações de entrada/saída de dados das tarefas e podem suspender uma execução quando for necessário, devolvendo o controle ao monitor.

INSTITUTO FEDERAL
RIO GRANDE DO SUL

SISTEMAS MULTI-TAREFAS

ESTADOS DAS TAREFAS



Várias tarefas podiam estar em andamento simultaneamente: uma estava ativa e as demais suspensas, esperando dados externos ou outras condições.

INSTITUTO FEDERAL
RIO GRANDE DO SUL

SISTEMAS MULTI-TAREFAS

Novo Problema:

E se uma tarefa não finalizasse? Se uma tarefa demorasse o resto da vida?

SISTEMAS DE TEMPO COMPARTILHADO

PROBLEMA DO LAÇO INFINITO: quando uma tarefa executar esse código, ela nunca encerrará nem será suspensa aguardando operações de entrada/saída de dados.

```
1  // calcula a soma dos primeiros 1000 inteiros
2
3  #include <stdio.h>
4
5  int main ()
6  {
7    int i = 0, soma = 0 ;
8
9    while (i <= 1000)
10     soma += i ;       // erro: o contador i não foi incrementado
11
12   printf ("A soma vale %d\n", soma);
13   exit(0) ;
14 }
```

SISTEMAS DE TEMPO COMPARTILHADO

SOLUÇÃO:

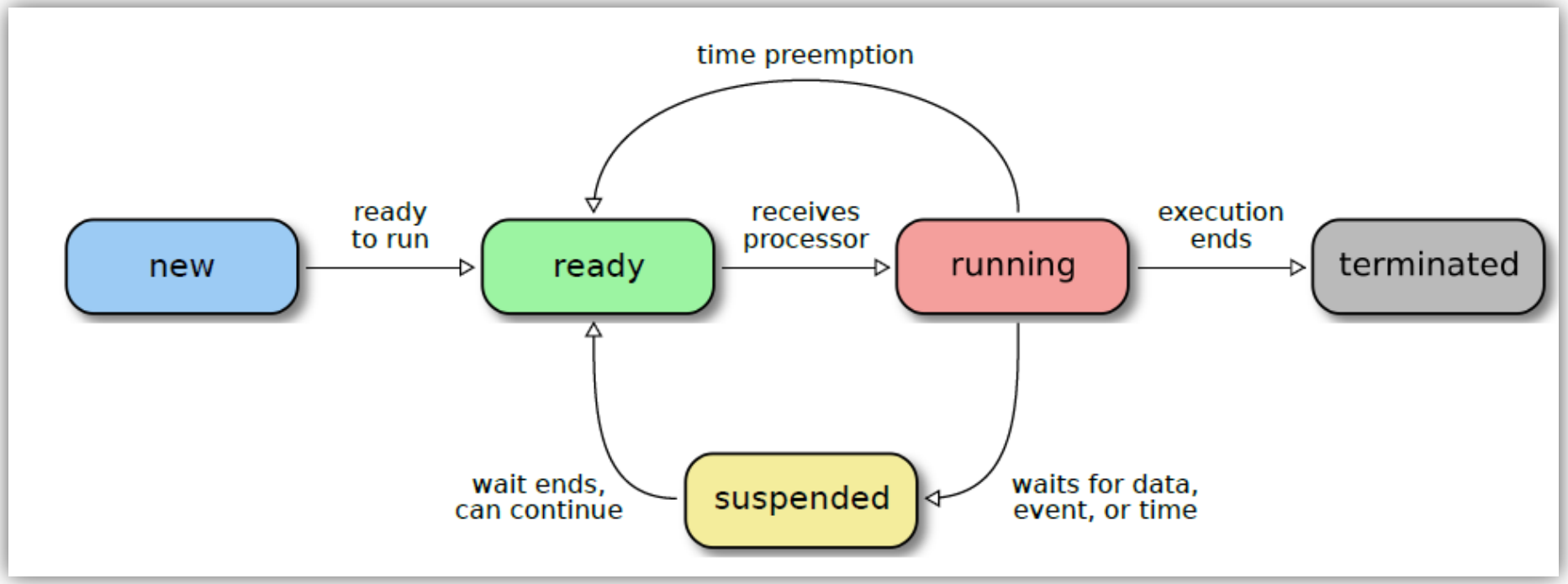
COMPARTILHAMENTO DE TEMPO

Para resolver essa questão, foi introduzido no início dos anos 60 um novo conceito: o compartilhamento de tempo, ou *time-sharing,* através do sistema **CTSS – Compatible Time-Sharing System**

SISTEMAS DE TEMPO COMPARTILHADO

- Nessa solução, para cada tarefa que recebe o processador é definido um prazo de processamento, denominado fatia de tempo ou quantum.
- Esgotado seu quantum, a tarefa em execução perde o processador e volta para uma fila de tarefas “prontas”, que estão na memória aguardando sua oportunidade de executar, e outra tarefa é ativada.
- O ato de retirar um recurso **“à força”** de uma tarefa (neste caso, o processador) **é denominado preempção**. Sistemas que implementam esse conceito são chamados **sistemas preemptivos.**
- Em um sistema operacional típico, a implementação da preempção por tempo usa as interrupções geradas por um temporizador programável disponível no hardware.

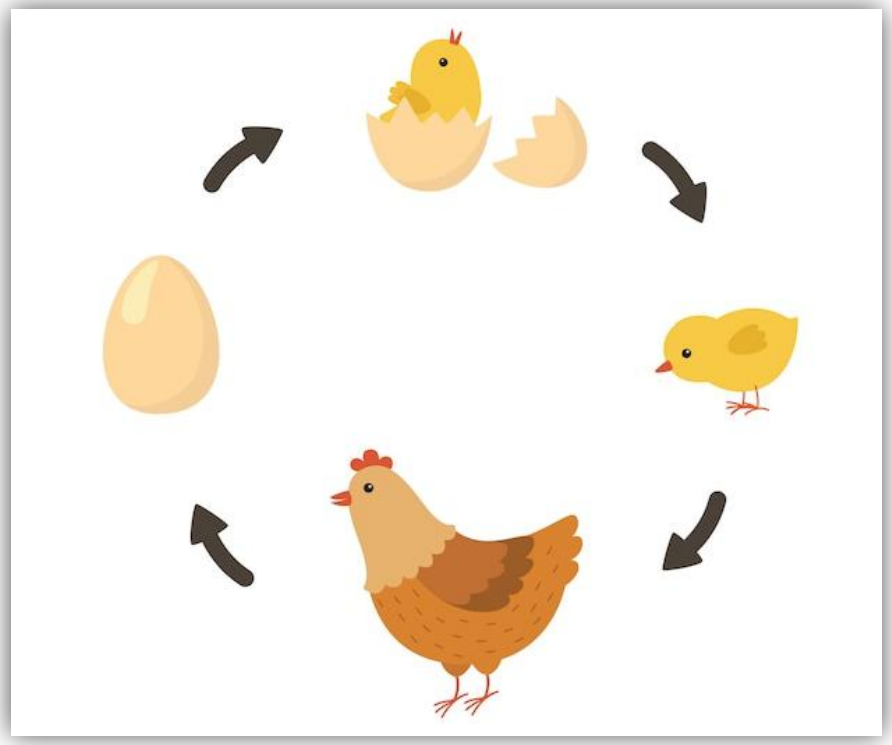
SISTEMAS DE TEMPO COMPARTILHADO



Cada tarefa recebe um quantum de tempo.
Quando o tempo terminar (Fim do quantum), perde o processador.
Implementado em hardware:

- Interrupção em tempo regulares
- Tratador de interrupção: move a tarefa para a lista de prontas.

Ciclo de vida das tarefas



Ciclo de vida das tarefas

Os estados e transições do ciclo de vida têm o seguinte significado:

- **Nova** : A tarefa está sendo criada.
  - Ex. eu código está sendo carregado em memória, junto com as bibliotecas necessárias, e as estruturas de dados do núcleo estão sendo atualizadas para permitir sua execução.

- **Pronta** : A tarefa está em memória, pronta para executar (ou para continuar sua execução), apenas aguardando a disponibilidade do processador.
  - Todas as tarefas prontas são organizadas em uma fila cuja ordem é determinada por algoritmos de escalonamento.

- **Executando** : O processador está dedicado à tarefa, executando suas instruções e fazendo avançar seu estado.

- **Suspensa** : A tarefa não pode executar porque depende de dados externos ainda não disponíveis (do disco ou da rede, por exemplo), aguarda algum tipo de sincronização (o fim de outra tarefa ou a liberação de algum recurso compartilhado) ou simplesmente espera o tempo passar (em uma operação sleeping, por exemplo).

- **Terminada** : O processamento da tarefa foi encerrado e ela pode ser removida da memória do sistema.

Ciclo de Vida das Tarefas

Tão importantes quanto os estados das tarefas apresentados são as transições entre esses estados:

- **Nova → Pronta** : ocorre quando a nova tarefa termina de ser carregada em memória, juntamente com suas bibliotecas e dados, estando pronta para executar.
- **Pronta → Executando** : esta transição ocorre quando a tarefa é escolhida pelo escalonador para ser executada, dentre as demais tarefas prontas.
- **Executando → Pronta** : esta transição ocorre quando se esgota a fatia de tempo destinada à tarefa (ou seja, o fim do quantum); como nesse momento a tarefa não precisa de outros recursos além do processador, ela volta à fila de tarefas prontas, para esperar novamente o processador.
- **Executando → Terminada** : ocorre quando a tarefa encerra sua execução ou é abortada em consequência de algum erro (acesso inválido à memória, instrução ilegal, divisão por zero, etc.). Na maioria dos sistemas a tarefa que deseja encerrar avisa o sistema operacional através de uma chamada de sistema (no Linux é usada a chamada exit).
- **Terminada → ....** : Uma tarefa terminada é removida da memória e seus registros e estruturas de controle no núcleo são apagadas.
- **Executando → Suspensa** : caso a tarefa em execução solicite acesso a um recurso não disponível, como dados externos ou alguma sincronização, ela abandona o processador e fica suspensa até o recurso ficar disponível.
- **Suspensa → Pronta** : quando o recurso solicitado pela tarefa se torna disponível, ela pode voltar a executar, portanto volta ao estado de pronta.

INSTITUTO FEDERAL
RIO GRANDE DO SUL

Ciclo de Vida das Tarefas

- Cite um estado de tareda inválido?

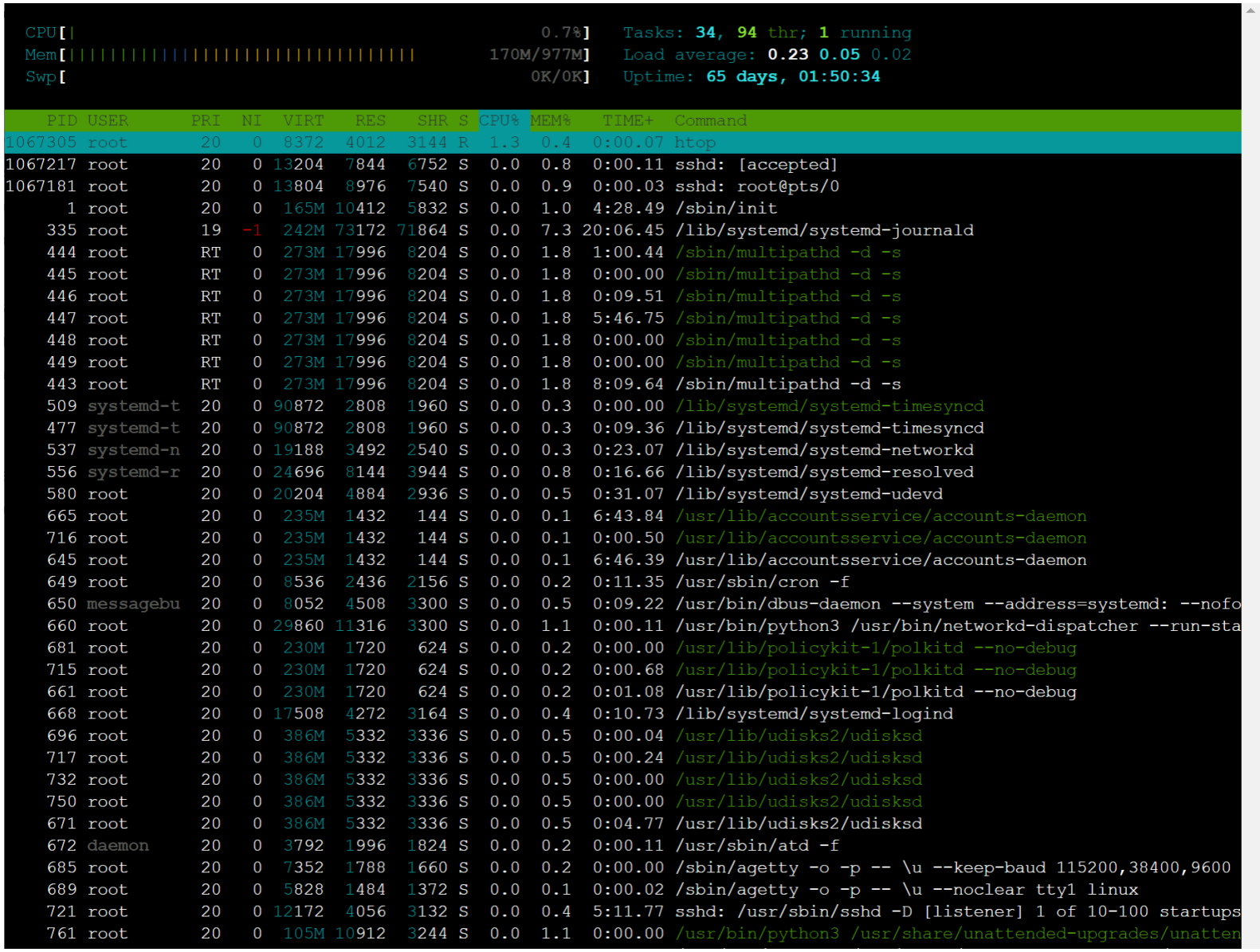
- Cite uma Transição de tarefa Inválida.

CONTEXTO: estado interno da tarefa

Cada tarefa possui um **TCB (Task Control Block)**

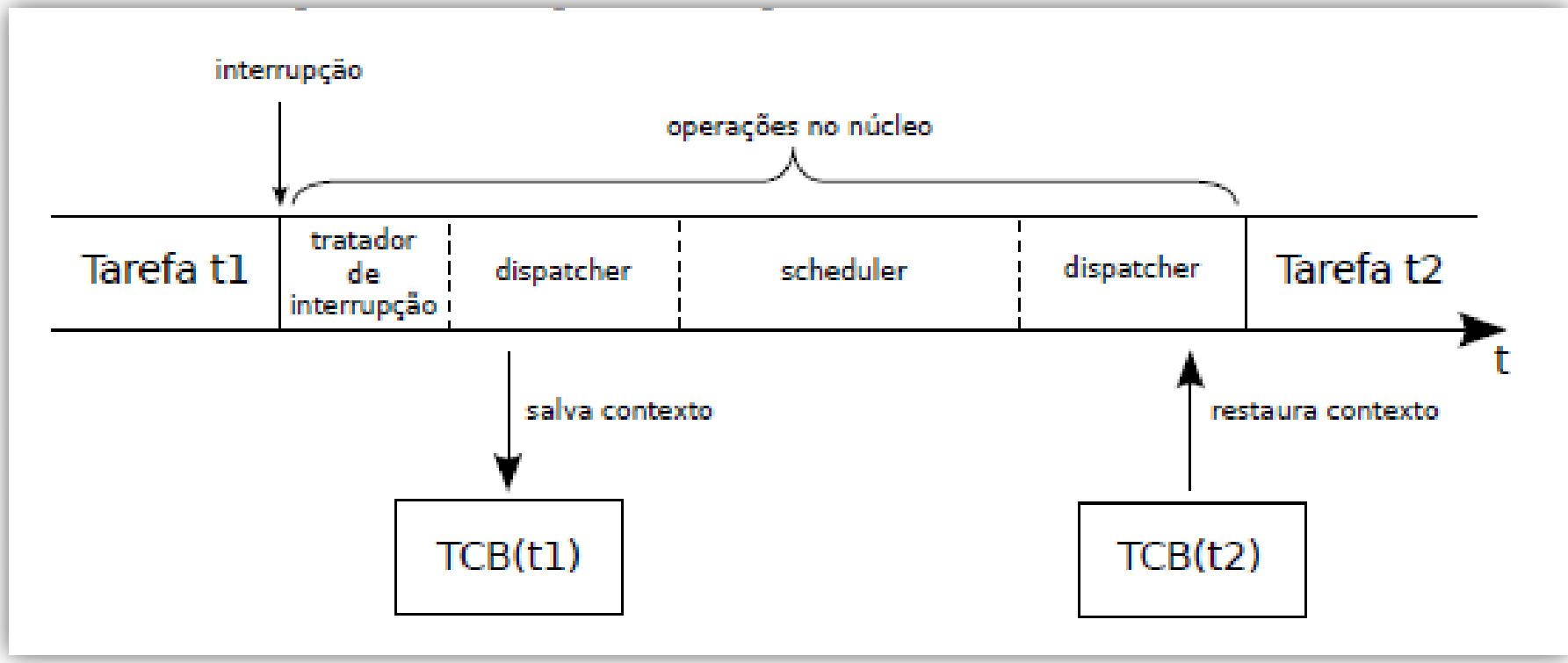
- Identificador da tarefa
- Estado da tarefa
- Informações de contexto do processador
  - (a posição de código (PC – program counter), os valores de suas variáveis, os arquivos, Stack Pointer (SP))
- Lista de áreas de memória usadas pela tarefa
- Recursos utilizados (Listas de arquivos abertos, conexões de rede, etc)
- Informações de gerência e contabilização (prioridade, usuário proprietário, data de início, tempo de processamento já decorrido, volume de dados lidos/escritos, etc.)

CONTEXTO: estado interno da tarefa



TROCA DE CONTEXTO

Necessária quando interrompe a execução de uma tarefa para retornar a ela mais tarde, sem corromper seu estado interno.



INSTITUTO FEDERAL
RIO GRANDE DO SUL

TROCA DE CONTEXTO

- **Dispatcher**: o armazenamento e recuperação do contexto e a atualização das informações contidas no TCB de cada tarefa são aspectos mecânicos, providos por um conjunto de rotinas denominado **despachante** ou **executivo**

- **Scheduler:** a escolha da próxima tarefa a receber o processador a cada troca de contexto é estratégica, podendo sofrer influências de diversos fatores, como as prioridades, os tempos de vida e os tempos de processamento restante de cada tarefa.

  - No Linux as operações de troca de contexto para a plataforma Intel x86 estão definidas através de diretivas em Assembly no arquivo **arch/i386/kernel/process.c** dos fontes do núcleo.

INSTITUTO FEDERAL
RIO GRANDE DO SUL

PROCESSOS

INSTITUTO FEDERAL
RIO GRANDE DO SUL

PROCESSOS

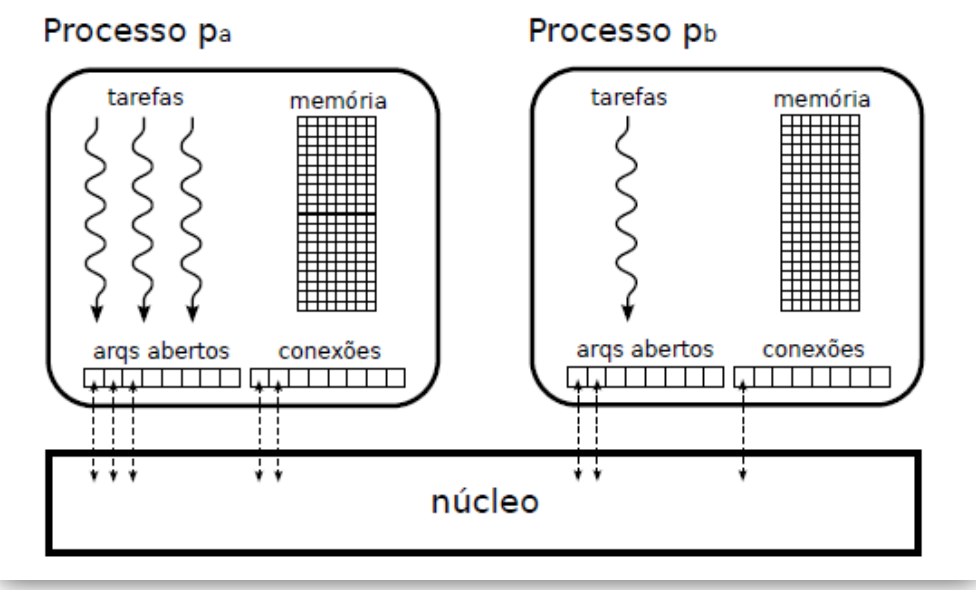
- **PROCESSOS são isolados entre si** pelos mecanismos de proteção providos pelo hardware (isolamento de áreas de memória, níveis de operação e chamadas de sistema) e pela própria gerência de tarefas, que atribui os recursos aos processos (e não às tarefas), impedindo que uma tarefa em execução no processo ***A*** acesse um recurso atribuído ao processo ***B***.

- **TCB SIMPLIFICADO:** a troca de contexto entre tarefas vinculadas ao mesmo processo é muito mais simples e rápida que entre tarefas vinculadas a processos distintos, pois somente os registradores do processador precisam ser salvos/restaurados (as áreas de memória e demais recursos são comuns às duas tarefas).

PROCESSOS

Uma unidade de contexto contém um PCB (***Process Control Blocks***):

- PID (Process ID)
- Cada TAREFA tem um TCB simplificado:
  - Identificador, Registradores, ref. ao processo



CRIAÇÃO DE PROCESSOS

Eventos que criam processos:

- Início do sistema.
- Execução de uma chamada de sistema de criação de processo por um processo em execução.
- Uma requisição do usuário para criar um novo processo.
- Início de uma tarefa em lote (batch job).

Classificação:

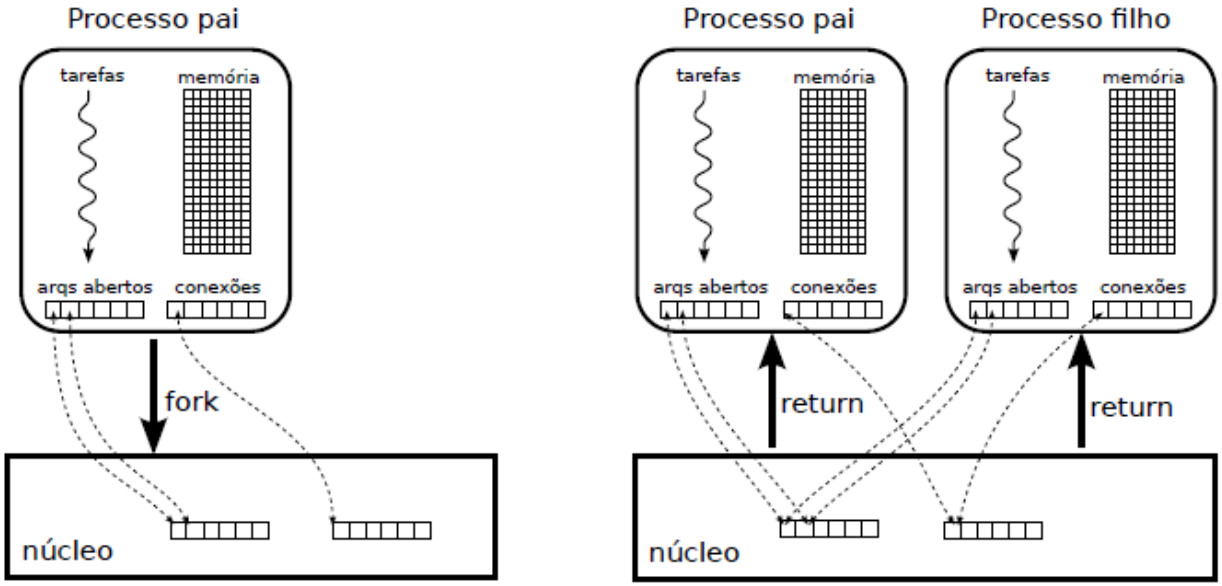
- **Foreground**: em primeiro plano
- **Background**: em segundo plano. Também chamados de **daemons**

CRIAÇÃO DE PROCESSOS

Feita por chamadas de sistemas.

- Exemplo no Unix:

fork → cria uma réplica do processo solicitante



INSTITUTO FEDERAL
RIO GRANDE DO SUL

CRIAÇÃO DE PROCESSOS

- **FORK**: todo o espaço de memória do processo é replicado, incluindo o código da(s) tarefa(s) associada(s) e os descritores dos arquivos e demais recursos associados ao mesmo.

- Ambos os processos têm os mesmos recursos associados, embora em áreas de memória distintas.

INSTITUTO FEDERAL
RIO GRANDE DO SUL

CRIAÇÃO DE PROCESSOS

```c
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[], char *envp[])
{
   int pid ;            /* identificador de processo */

   pid = fork () ;      /* replicação do processo */

   if ( pid < 0 )       /* fork não funcionou */
   {
      perror ("Erro: ") ;
      exit (-1) ;       /* encerra o processo */
   }
   else if ( pid > 0 )   /* sou o processo pai */
   {
      wait (0) ;        /* vou esperar meu filho concluir */
   }
   else                 /* sou o processo filho*/
   {
      /* carrega outro código binário para executar */
      execve ("/bin/date", argv, envp) ;
      perror ("Erro: ") ; /* execve não funcionou */
   }
   printf ("Tchau !\n") ;
   exit(0) ;            /* encerra o processo */
}
```

PROCESSOS Linux (FORK)

Um processo pode estar num dos seguintes estados:

- **idle/new:**

  Processo recém criado (sendo preparado).

- **ready:**

  Processo sendo carregado.

- **standby:**

  O próximo a ser executado.

- **running:**

  O processo esta rodando (ativo).

- **blocked:**

  Esperando evento externo (entrada do usuário) (inativo).

- **suspended-blocked:**

  Processo suspenso.

- **zoombied:**

  O processo acabou mas não informou seu pai.

- **done-terminated:**

  O processo foi encerrado e os recursos liberados.

INSTITUTO FEDERAL
RIO GRANDE DO SUL

TÉRMINO DE PROCESSOS

Eventos que terminam processos:

- Saída normal (voluntária)
- Saída por erro (voluntária)
- Erro fatal (involuntário)
- Cancelamento por um outro processo (involuntário)

Comandos:

- Unix: exit
- Windows: ExitProcess

INSTITUTO FEDERAL
RIO GRANDE DO SUL

CRIAÇÃO DE PROCESSOS

Processos sincronizados e não sincronizados:

• Quando o processo pai espera o encerramento do processo filho, ele tem uma execução sincronizada.
• Quando os processos rodam independentemente eles são não sincronizados.

A comunicação entre processos pode ser realizada utilizando-se variáveis de ambiente, pipes, ou memória compartilhada (shared memory).

THREADS

INSTITUTO FEDERAL
RIO GRANDE DO SUL

Threads

O que são threads ?

Threads são múltiplos caminhos de execução que rodam concorrentemente na memória compartilhada e que compartilham os mesmos recursos e sinais do processo pai. Uma thread é um processo simplificado, mais leve ou “light”, custa pouco para o sistema operacional, sendo fácil de criar, manter e gerenciar.

Casos em que o uso de threads é interessante:

* Para salvar arquivos em disco.
* Quando a interface gráfica é pesada.
* Quando existem comunicações pela internet.
* Quando existem cálculos pesados.

INSTITUTO FEDERAL
RIO GRANDE DO SUL

Threads

Conceito

- É um fluxo de execução associado ao processo.

- Tipo:
  - User thread
  - Kernel thread

Vantagens

- São mais fáceis de implementar.
- Sobreposição de tarefas na mesma aplicação.

INSTITUTO FEDERAL
RIO GRANDE DO SUL

Threads

- **USER THREAD:** Threads executando dentro de um processo são chamados de **threads de usuário** (user-level threads ou simplesmente user threads). Cada thread de usuário corresponde a uma tarefa a ser executada dentro de um processo.

- **KERNEL THREAD:** os fluxos de execução reconhecidos e gerenciados pelo núcleo do sistema operacional são chamados de **threads de núcleo** (kernel-level threads ou kernel threads).

Sobreposição de tarefas na mesma aplicação (bom para app com mt e/s)

INSTITUTO FEDERAL
RIO GRANDE DO SUL

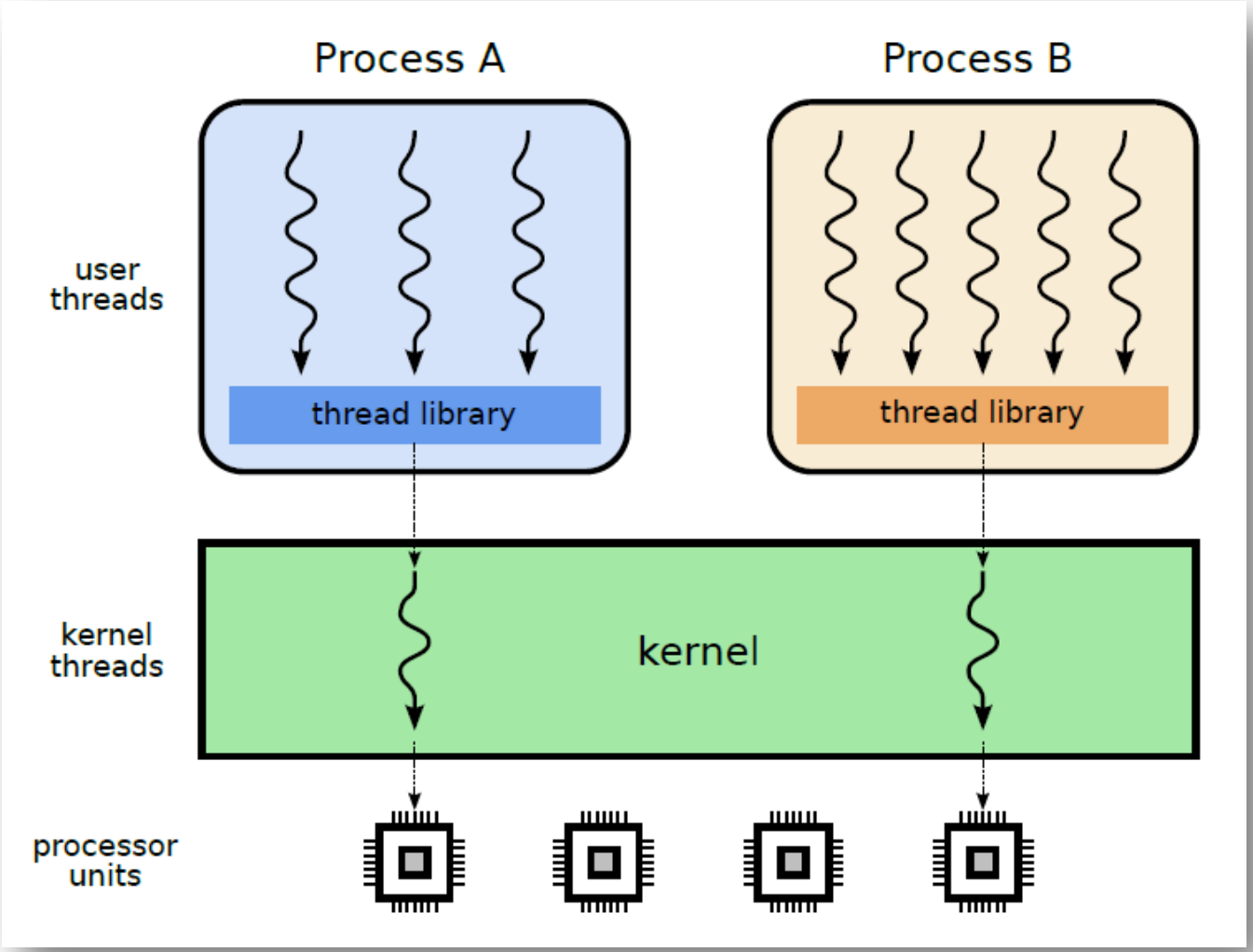
Modelo de threads N:1

- Os sistemas operacionais mais antigos suportavam apenas processos sequenciais, com um único fluxo de execução em cada um. Os desenvolvedores de aplicações contornaram esse problema construindo bibliotecas para salvar, modificar e restaurar os registradores da CPU dentro do processo, permitindo assim criar e gerenciar vários fluxos de execução (threads) dentro de cada processo, sem a participação do núcleo.

- Com essas bibliotecas, uma aplicação pode lançar várias threads conforme sua necessidade, mas o núcleo do sistema irá sempre perceber (e gerenciar) apenas um fluxo de execução dentro de cada processo (ou seja, o núcleo irá manter apenas uma thread de núcleo por processo). Esta forma de implementação de threads é denominada **Modelo de Threads N:1, pois N threads dentro de um processo são mapeadas em uma única** thread no núcleo.

INSTITUTO FEDERAL
RIO GRANDE DO SUL

Modelo de threads N:1



INSTITUTO FEDERAL
RIO GRANDE DO SUL

Modelo de threads 1:1

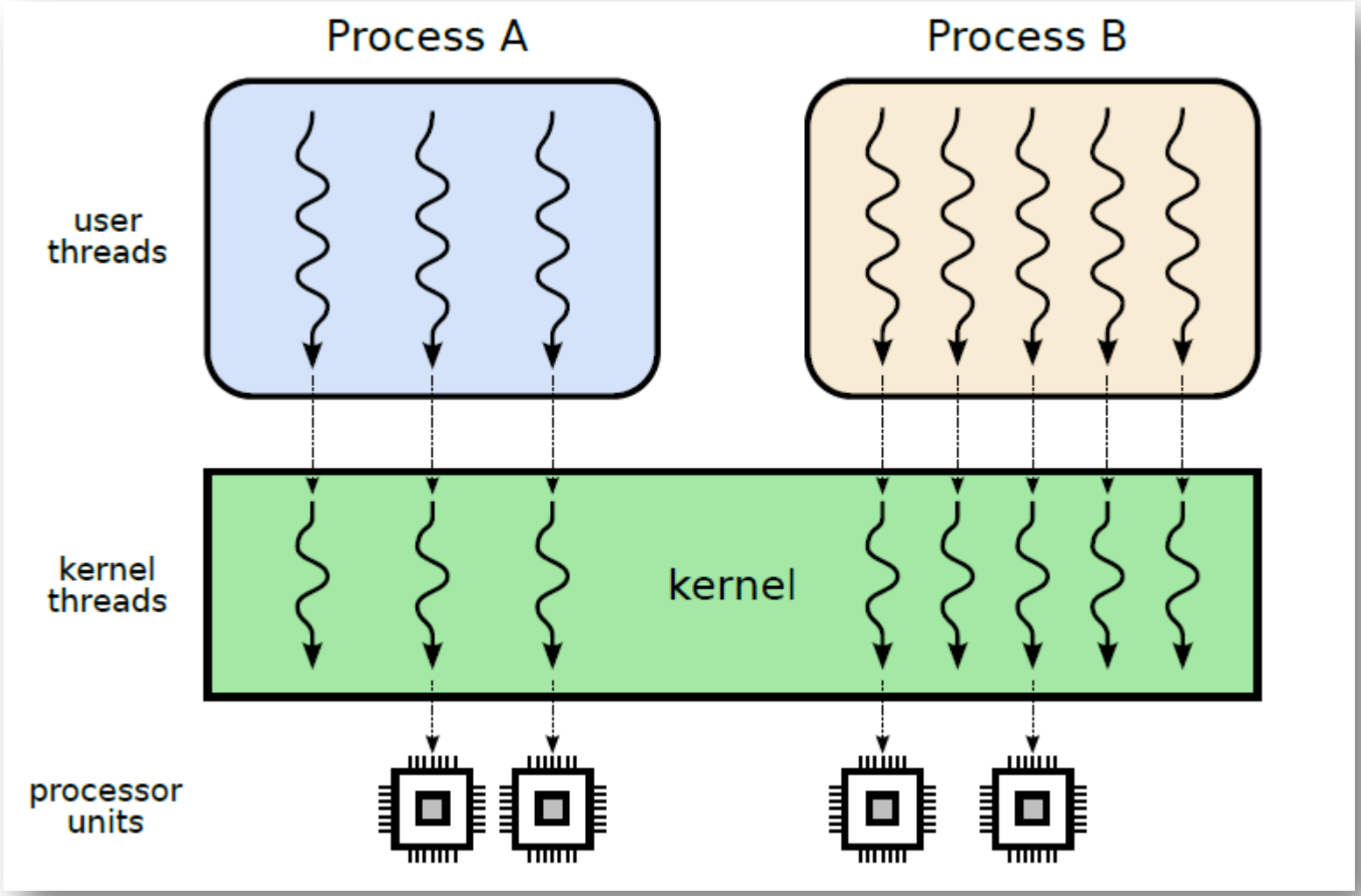
Para cada thread de usuário foi então associado um thread correspondente dentro do núcleo, suprimindo com isso a **necessidade de bibliotecas de threads.**

Este é o modelo mais frequente nos sistemas operacionais atuais, como o Windows NT e seus descendentes, além da maioria dos UNIXes.

Este modelo é pouco escalável: a criação de um número muito grande de threads impõe uma carga elevada ao núcleo do sistema, inviabilizando aplicações com muitas tarefas (como grandes servidores Web e simulações de grande porte).

INSTITUTO FEDERAL
RIO GRANDE DO SUL

Modelo de threads 1:1



INSTITUTO FEDERAL
RIO GRANDE DO SUL

Modelo de threads N:M

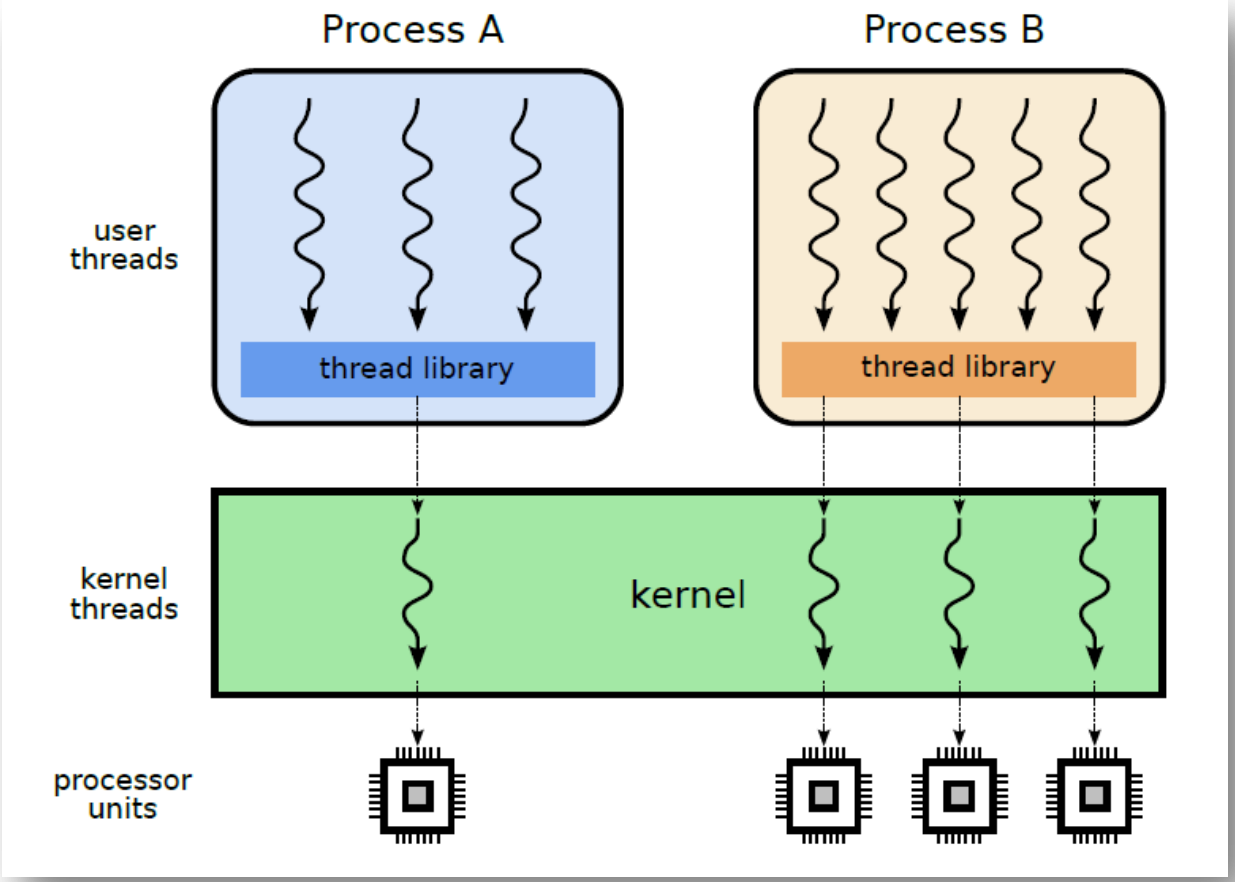
Para resolver o problema de escalabilidade da abordagem 1:1, alguns sistemas operacionais implementam um modelo híbrido, que agrega características dos modelos anteriores.

Nesse novo modelo, uma biblioteca gerencia um conjunto de N threads de usuário (dentro do processo), que é mapeado em M < N threads no núcleo

O modelo N:M é implementado pelo Solaris e também pelo projeto KSE (Kernel-Scheduled Entities) do FreeBSD.

INSTITUTO FEDERAL
RIO GRANDE DO SUL

Modelo de threads N:M



THREAD POSIX

- PADRÃO IEEE 1003.1c
  - Cada sistema operacional implementava sua própria versão de threads, Para superar esse problema foi desenvolvido o padrão POSIX 1003.1.c (Portabel Operating Sysrtem Interface) pelo IEEE (Institute of Electrical and Electronics Engineers)
- +60 FUNÇÕES

Chamada de thread	Descrição
pthread_create	Cria um novo thread
pthread_exit	Conclui a chamada de thread
pthread_join	Espera que um thread específico seja abandonado
pthread_yield	Libera a CPU para que outro thread seja executado
pthread_attr_init	Cria e inicializa uma estrutura de atributos do thread
pthread_attr_destroy	Remove uma estrutura de atributos do thread

INSTITUTO FEDERAL
RIO GRANDE DO SUL

Threads

Existem diferentes categorias de sincronização, dentre as quais pode-se destacar:

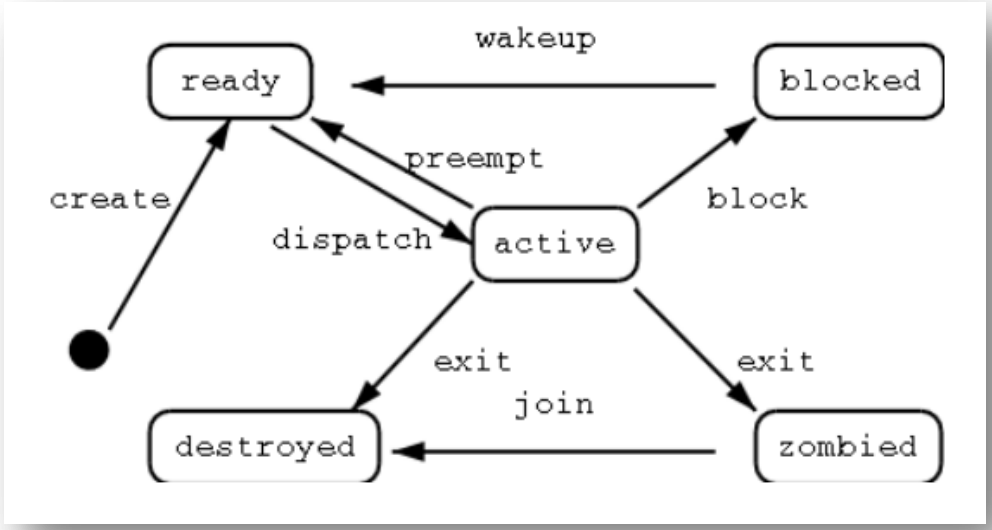
Sincronização de dados: Threads concorrendo no acesso de uma variável compartilhada ou a arquivos compartilhados.

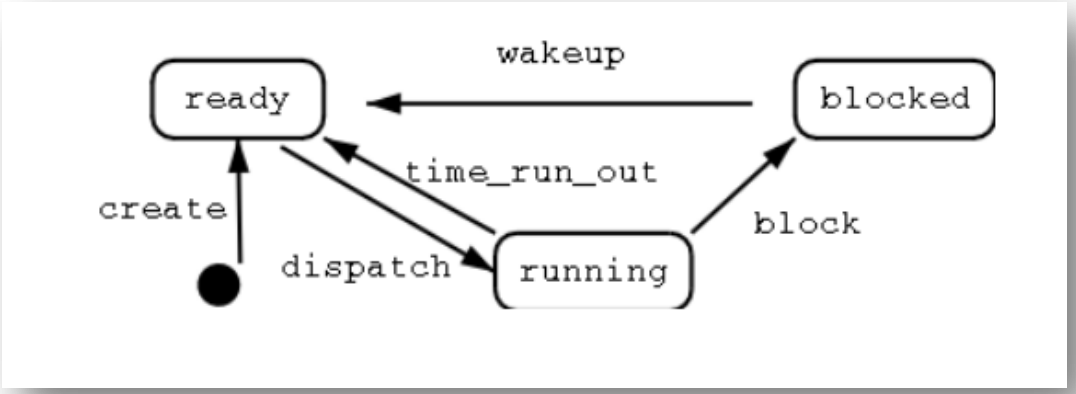
Sincronização de hardware: Acesso compartilhado a tela, impressora.

Sincronização de tarefas: Condições lógicas

INSTITUTO FEDERAL
RIO GRANDE DO SUL

Processos X Threads





INSTITUTO FEDERAL
RIO GRANDE DO SUL

Criando Threads

```
int pthread_create(pthread_t *tid,
                   const pthread_attr_t *attr,
                   void * (* start)(void *),
                   void *arg);
```

- 1) Cada novo thread é representado por um identificador (thread identifier ou tid) de tipo pthread_t.
- 2) O segundo parâmetro serve para indicar uma série de atributos e propriedades que o novo thread deverá ter.
- 3) O terceiro parâmetro indica a função de início do thread.
- 4) Finalmente o quarto parâmetro é o valor do argumento a ser passado à função de início, como seu parâmetro.
  - **Uma thread retorna 0 no caso de sucesso e um código de erro no caso contrário.**

INSTITUTO FEDERAL
RIO GRANDE DO SUL

Finalizando Threads

- Um thread termina quando a função de início, indicada quando da criação, retornar, ou quando o próprio thread invocar o serviço de terminação:

```
pthread_exit(void *value_ptr);
```

Onde **value_ptr** é o ponteiro que o thread deve ter como resultado.

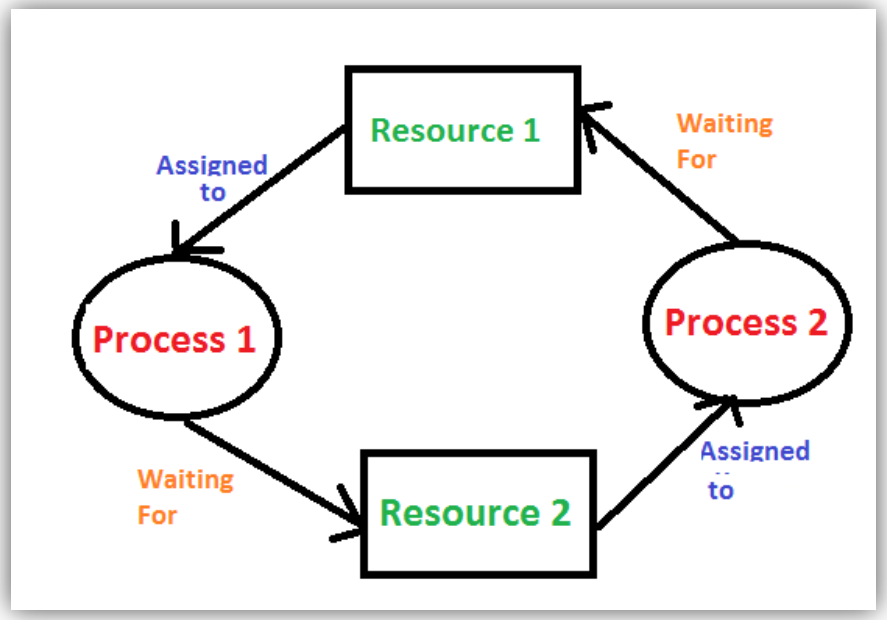
Sincronização de Threads

```
int  pthread_join (pthread_t thread,
                   void ** status);
```

- O comando join é executado para esperar o fim de uma Thread , de maneira semelhante a wait( ) para processos (bloqueia o thread que *chamou* esta função até que o thread com o identificador *thread_id* termine.).
  - O Identificador da thread deve ser especificado.

Sincronização de Threads

- Deadlock ocorre quando aqcontece um **impasse**, e dois ou mais processos ficam impedidos de continuar suas execuções, ou seja, ficam bloqueados, esperando uns pelos outros.



Criação de Threads