

# SO: Multitarefa e Processos

## Projetos de Sistemas Operacionais

Prof. Dr. Denis M. L. Martins


Engenharia de Computação: 5º Semestre

# Introdução

- Explicar conceitos e relações entre programa, processo, thread e multitarefa.
- Compreender a diferenças entre threads e processos.
- Entender os estados de uma processo.

Parte do material apresentado a seguir foi adaptado de *IT Systems – Open Educational Resource*, disponível em <https://oer.gitlab.io/oer-courses/it-systems/>, produzido por [Jens Lechtenböger](#), e distribuído sob a licença [CC BY-SA 4.0](#).

# Multitarefa

- Os sistemas operacionais permitem que múltiplas computações ocorram **concorrentemente** em um único sistema computacional.
  - ▶ Um exemplo de **multitarefa** ocorre quando você está **ouvindo música no Spotify** enquanto **navega na internet e clica em links**.
  - ▶ O **sistema operacional** gerencia a execução simultânea do player de música e do navegador.
- Para isso, o sistema:
  - ▶ Divide o tempo (*time slicing*) do hardware entre os diferentes operações em execução (via **Escalonamento**).
  - ▶ Gerencia as transições entre as operações.
  - ▶ Mantém o controle do estado de cada operação para que possam ser retomados corretamente.
- **Paralelismo versus Concorrência** .

- Mesmo em um único CPU core: ilusão de simultaneidade (computação "paralela").
- Essa capacidade é essencial para:
  - ▶ Garantir **eficiência** no uso dos recursos computacionais.
  - ▶ Proporcionar **responsividade**, permitindo que múltiplos programas rodem de forma contínua e sem atrasos perceptíveis.
  - ▶ Melhorar a **utilização do sistema**, possibilitando a execução simultânea de várias tarefas.

## ● Conceitos de Threads e Processos

- ▶ Diferentes termos podem ser usados para se referir a execuções ("computações") em um computador: *threads*, processos, tarefas ou trabalhos.
- ▶ Neste contexto, os termos **thread** e **processo** possuem significados distintos.

## ● Thread

- ▶ Unidade fundamental de concorrência.
- ▶ Representa uma sequência de ações programadas (executadas em um CPU core).
- ▶ Um programa pode criar múltiplas threads para executar diferentes tarefas simultaneamente.
- ▶ Mesmo que um programa crie apenas uma thread, um sistema típico executa diversas threads simultaneamente, incluindo aquelas do sistema operacional.

## ● Processo

- ▶ Criado sempre que um programa é iniciado.
- ▶ Atua como um contêiner que gerencia e protege uma ou mais threads.
- ▶ Impede que threads de processos diferentes interfiram umas nas outras.
- ▶ Exemplo: uma thread em um processo não pode sobrescrever a memória de outro processo.



## ● Conceitos de Threads e Processos

- ▶ Diferentes termos podem ser usados para se referir a execuções ("computações") em um computador: *threads*, processos, tarefas ou trabalhos.
- ▶ Neste contexto, os termos **thread** e **processo** possuem significados distintos.

## ● Thread

- ▶ Unidade fundamental de concorrência.
- ▶ Representa uma sequência de ações programadas (executadas em um CPU core).
- ▶ Um programa pode criar múltiplas threads para executar diferentes tarefas simultaneamente.
- ▶ Mesmo que um programa crie apenas uma thread, um sistema típico executa diversas threads simultaneamente, incluindo aquelas do sistema operacional.

## ● Processo

- ▶ Criado sempre que um programa é iniciado.
- ▶ Atua como um contêiner que gerencia e protege uma ou mais threads.
- ▶ Impede que threads de processos diferentes interfiram umas nas outras.
- ▶ Exemplo: uma thread em um processo não pode sobrescrever a memória de outro processo.

## ● Conceitos de Threads e Processos

- ▶ Diferentes termos podem ser usados para se referir a execuções ("computações") em um computador: *threads*, processos, tarefas ou trabalhos.
- ▶ Neste contexto, os termos **thread** e **processo** possuem significados distintos.

## ● Thread

- ▶ Unidade fundamental de concorrência.
- ▶ Representa uma sequência de ações programadas (executadas em um CPU core).
- ▶ Um programa pode criar múltiplas threads para executar diferentes tarefas simultaneamente.
- ▶ Mesmo que um programa crie apenas uma thread, um sistema típico executa diversas threads simultaneamente, incluindo aquelas do sistema operacional.

## ● Processo

- ▶ Criado sempre que um programa é iniciado.
- ▶ Atua como um contêiner que gerencia e protege uma ou mais threads.
- ▶ Impede que threads de processos diferentes interfiram umas nas outras.
- ▶ Exemplo: uma thread em um processo não pode sobrescrever a memória de outro processo.

## ● Conceitos de Threads e Processos

- ▶ Diferentes termos podem ser usados para se referir a execuções ("computações") em um computador: *threads*, processos, tarefas ou trabalhos.
- ▶ Neste contexto, os termos **thread** e **processo** possuem significados distintos.

## ● Thread

- ▶ Unidade fundamental de concorrência.
- ▶ Representa uma sequência de ações programadas (executadas em um CPU core).
- ▶ Um programa pode criar múltiplas threads para executar diferentes tarefas simultaneamente.
- ▶ Mesmo que um programa crie apenas uma thread, um sistema típico executa diversas threads simultaneamente, incluindo aquelas do sistema operacional.

## ● Processo

- ▶ Criado sempre que um programa é iniciado.
- ▶ Atua como um contêiner que gerencia e protege uma ou mais threads.
- ▶ Impede que threads de processos diferentes interfiram umas nas outras.
- ▶ Exemplo: uma thread em um processo não pode sobrescrever a memória de outro processo.

## ● Conceitos de Threads e Processos

- ▶ Diferentes termos podem ser usados para se referir a execuções ("computações") em um computador: *threads*, processos, tarefas ou trabalhos.
- ▶ Neste contexto, os termos **thread** e **processo** possuem significados distintos.

## ● Thread

- ▶ Unidade fundamental de concorrência.
- ▶ Representa uma sequência de ações programadas (executadas em um CPU core).
- ▶ Um programa pode criar múltiplas threads para executar diferentes tarefas simultaneamente.
- ▶ Mesmo que um programa crie apenas uma thread, um sistema típico executa diversas threads simultaneamente, incluindo aquelas do sistema operacional.

## ● Processo

- ▶ Criado sempre que um programa é iniciado.
- ▶ Atua como um contêiner que gerencia e protege uma ou mais threads.
- ▶ Impede que threads de processos diferentes interfiram umas nas outras.
- ▶ Exemplo: uma thread em um processo não pode sobrescrever a memória de outro processo.

## ● Conceitos de Threads e Processos

- ▶ Diferentes termos podem ser usados para se referir a execuções ("computações") em um computador: *threads*, processos, tarefas ou trabalhos.
- ▶ Neste contexto, os termos **thread** e **processo** possuem significados distintos.

## ● Thread

- ▶ Unidade fundamental de concorrência.
- ▶ Representa uma sequência de ações programadas (executadas em um CPU core).
- ▶ Um programa pode criar múltiplas threads para executar diferentes tarefas simultaneamente.
- ▶ Mesmo que um programa crie apenas uma thread, um sistema típico executa diversas threads simultaneamente, incluindo aquelas do sistema operacional.

## ● Processo

- ▶ Criado sempre que um programa é iniciado.
- ▶ Atua como um contêiner que gerencia e protege uma ou mais threads.
- ▶ Impede que threads de processos diferentes interfiram umas nas outras.
- ▶ Exemplo: uma thread em um processo não pode sobrescrever a memória de outro processo.

## ● Conceitos de Threads e Processos

- ▶ Diferentes termos podem ser usados para se referir a execuções ("computações") em um computador: *threads*, processos, tarefas ou trabalhos.
- ▶ Neste contexto, os termos **thread** e **processo** possuem significados distintos.

## ● Thread

- ▶ Unidade fundamental de concorrência.
- ▶ Representa uma sequência de ações programadas (executadas em um CPU core).
- ▶ Um programa pode criar múltiplas threads para executar diferentes tarefas simultaneamente.
- ▶ Mesmo que um programa crie apenas uma thread, um sistema típico executa diversas threads simultaneamente, incluindo aquelas do sistema operacional.

## ● Processo

- ▶ Criado sempre que um programa é iniciado.
- ▶ Atua como um contêiner que gerencia e protege uma ou mais threads.
- ▶ Impede que threads de processos diferentes interfiram umas nas outras.
- ▶ Exemplo: uma thread em um processo não pode sobrescrever a memória de outro processo.

## ● Conceitos de Threads e Processos

- ▶ Diferentes termos podem ser usados para se referir a execuções ("computações") em um computador: *threads*, processos, tarefas ou trabalhos.
- ▶ Neste contexto, os termos **thread** e **processo** possuem significados distintos.

## ● Thread

- ▶ Unidade fundamental de concorrência.
- ▶ Representa uma sequência de ações programadas (executadas em um CPU core).
- ▶ Um programa pode criar múltiplas threads para executar diferentes tarefas simultaneamente.
- ▶ Mesmo que um programa crie apenas uma thread, um sistema típico executa diversas threads simultaneamente, incluindo aquelas do sistema operacional.

## ● Processo

- ▶ Criado sempre que um programa é iniciado.
- ▶ Atua como um contêiner que gerencia e protege uma ou mais threads.
- ▶ Impede que threads de processos diferentes interfiram umas nas outras.
- ▶ Exemplo: uma thread em um processo não pode sobrescrever a memória de outro processo.

## ● Conceitos de Threads e Processos

- ▶ Diferentes termos podem ser usados para se referir a execuções ("computações") em um computador: *threads*, processos, tarefas ou trabalhos.
- ▶ Neste contexto, os termos **thread** e **processo** possuem significados distintos.

## ● Thread

- ▶ Unidade fundamental de concorrência.
- ▶ Representa uma sequência de ações programadas (executadas em um CPU core).
- ▶ Um programa pode criar múltiplas threads para executar diferentes tarefas simultaneamente.
- ▶ Mesmo que um programa crie apenas uma thread, um sistema típico executa diversas threads simultaneamente, incluindo aquelas do sistema operacional.

## ● Processo

- ▶ Criado sempre que um programa é iniciado.
- ▶ Atua como um contêiner que gerencia e protege uma ou mais threads.
- ▶ Impede que threads de processos diferentes interfiram umas nas outras.
- ▶ Exemplo: uma thread em um processo não pode sobrescrever a memória de outro processo.



## ● Conceitos de Threads e Processos

- ▶ Diferentes termos podem ser usados para se referir a execuções ("computações") em um computador: *threads*, processos, tarefas ou trabalhos.
- ▶ Neste contexto, os termos **thread** e **processo** possuem significados distintos.

## ● Thread

- ▶ Unidade fundamental de concorrência.
- ▶ Representa uma sequência de ações programadas (executadas em um CPU core).
- ▶ Um programa pode criar múltiplas threads para executar diferentes tarefas simultaneamente.
- ▶ Mesmo que um programa crie apenas uma thread, um sistema típico executa diversas threads simultaneamente, incluindo aquelas do sistema operacional.

## ● Processo

- ▶ Criado sempre que um programa é iniciado.
- ▶ Atua como um contêiner que gerencia e protege uma ou mais threads.
- ▶ Impede que threads de processos diferentes interfiram umas nas outras.
- ▶ Exemplo: uma thread em um processo não pode sobrescrever a memória de outro processo.

# Processos

- Processo  $\approx$  programa em execução.
  - ▶ **Programa:** entidade **passiva** guardada no disco (**arquivo executável**).
  - ▶ **Processo:** carregado em memória.
- Um programa pode criar múltiplos processos, e.g., um processo por tab aberta no navegador.
  - ▶ No Firefox, digite `about:processes` na barra de endereço.
- Relacionamento *many-to-many* entre **programas** e **processos** 🖥️.
- Processo como unidade de **gerenciamento** e **proteção**.
- O que há em um processo? (Ver próximo slide)



Figura 1: O que há em um processo?

Créditos: [Julia Evans](#).

# Bloco de Controle de Processo (PCB)

Representação de um processo no SO:

- **Estado do processo:** em execução, em espera, etc.
- **Contador de programa (PC):** endereço da próxima instrução.
- **Registadores da CPU:** conteúdo de todos os registradores utilizados por processos.
- **Informações de escalonamento:** prioridades, ponteiros para filas.
- **Informação de gerenciamento de memória:** memória alocada para o processo.
- **Estatísticas:** uso de CPU, tempo desde o início, limites de tempo.
- **Informações de I/O:** dispositivos alocados ao processo, lista de arquivos abertos.

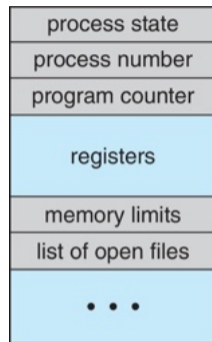


Figura 2: Bloco de Controle de Processo, do inglês *Process Control Block*. Créditos: Silberschatz, Galvin and Gagne, 2018.

# Layout de Memória de um programa em C

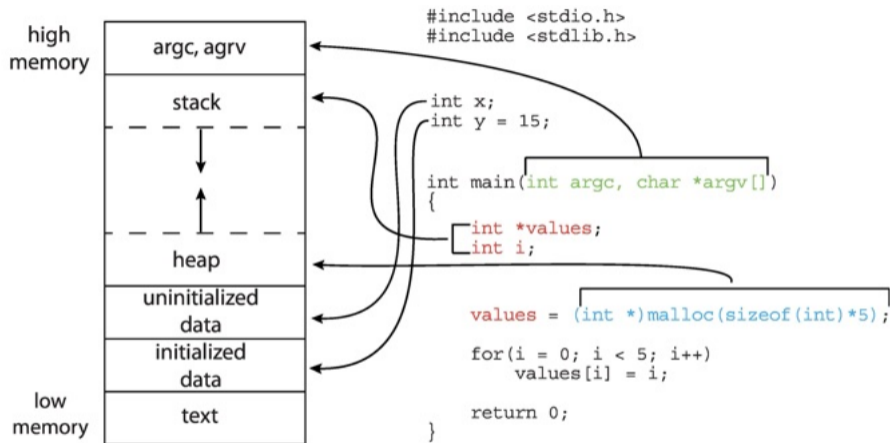



Figura 3: Layout de um programa escrito em Linguagem C na memória.  
Créditos: Silberschatz, Galvin and Gagne, 2018.

Um processo na memória respeita o seguinte layout :

- Código do programa: **text section**.
- Atividade atual: **program counter** (PC), registradores.
- **Stack** (pilha) contendo dados temporários.
- Parâmetros de funções, endereços de retorno, variáveis locais.
- **Data section** contendo variáveis globais.
- **Heap** contendo memória alocada dinamicamente.

Conforme um processo executa, ele muda de estado:

- **Novo:** O processo está sendo criado.
- **Em execução:** Instruções estão sendo executadas.
- **Em espera:** O processo está esperando por um evento.
- **Pronto:** O processo está esperando que seja atribuído a um processador.
- **Concluído:** O processo terminou sua execução.

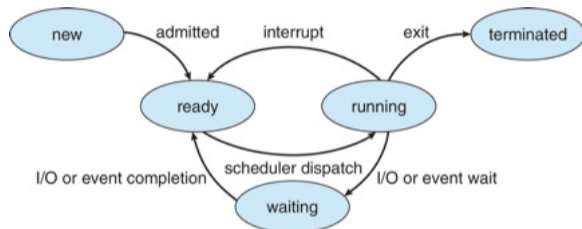


Figura 4: Diagrama de estados de um processo. Créditos: Silberschatz, Galvin and Gagne, 2018.

Conforme um processo executa, ele muda de estado:

- **Novo:** O processo está sendo criado.
- **Em execução:** Instruções estão sendo executadas.
- **Em espera:** O processo está esperando por um evento.
- **Pronto:** O processo está esperando que seja atribuído a um processador.
- **Concluído:** O processo terminou sua execução.

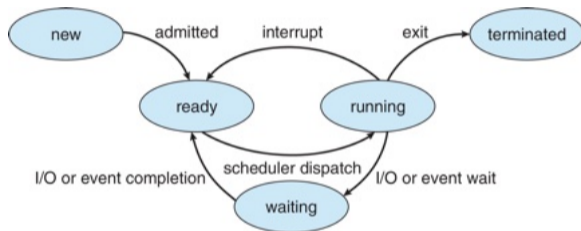


Figura 4: Diagrama de estados de um processo.  
Créditos: Silberschatz, Galvin and Gagne, 2018.



Conforme um processo executa, ele muda de estado:

- **Novo:** O processo está sendo criado.
- **Em execução:** Instruções estão sendo executadas.
- **Em espera:** O processo está esperando por um evento.
- **Pronto:** O processo está esperando que seja atribuído a um processador.
- **Concluído:** O processo terminou sua execução.

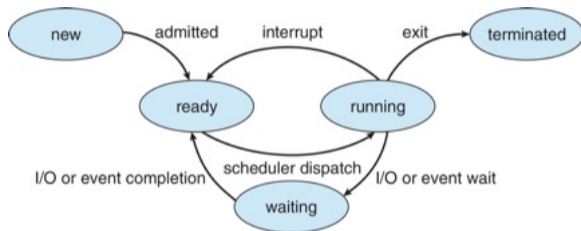


Figura 4: Diagrama de estados de um processo.  
Créditos: Silberschatz, Galvin and Gagne, 2018.

Conforme um processo executa, ele muda de estado:

- **Novo:** O processo está sendo criado.
- **Em execução:** Instruções estão sendo executadas.
- **Em espera:** O processo está esperando por um evento.
- **Pronto:** O processo está esperando que seja atribuído a um processador.
- **Concluído:** O processo terminou sua execução.

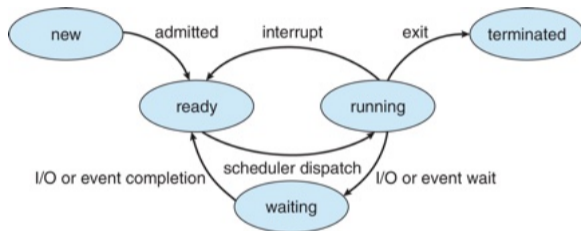


Figura 4: Diagrama de estados de um processo. Créditos: Silberschatz, Galvin and Gagne, 2018.

Conforme um processo executa, ele muda de estado:

- **Novo:** O processo está sendo criado.
- **Em execução:** Instruções estão sendo executadas.
- **Em espera:** O processo está esperando por um evento.
- **Pronto:** O processo está esperando que seja atribuído a um processador.
- **Concluído:** O processo terminou sua execução.

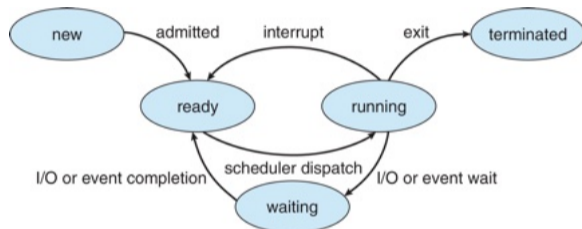
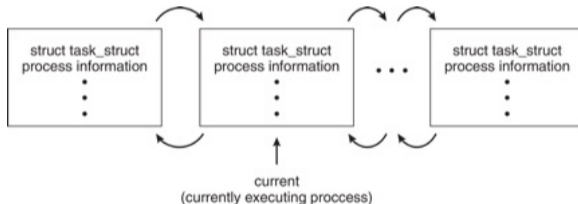


Figura 4: Diagrama de estados de um processo.  
Créditos: Silberschatz, Galvin and Gagne, 2018.

# Representação de um Processo em Linux

```
pid t_pid;                /* process identifier */
long state;               /* state of the process */
unsigned int time_slice  /* scheduling information */
struct task_struct *parent; /* this process's parent */
struct list_head children; /* this process's children */
struct files_struct *files; /* list of open files */
struct mm_struct *mm;     /* address space of this process */
```



**Figura 5:** Lista duplamente encadeada representando processos ativos no Linux.  
Créditos: Silberschatz, Galvin and Gagne, 2018.

- O kernel guarda informações sobre processos no diretório /proc.
- Não armazena arquivos reais, mas contém entradas geradas dinamicamente pelo kernel.
- Facilita o monitoramento de processos (/proc/PID/ contém detalhes sobre um processo específico).
- Verificação de informações do sistema (/proc/cpuinfo, /proc/meminfo).
- O comando ps inspeciona /proc.

## an amazing directory: /proc JULIA EVANS @b0rk


Every process on Linux has a PID (process ID) like 42.  In /proc/42, there's a lot of VERY USEFUL information about process 42	<b>/proc/PID/cmdline</b> command line arguments the process was started with	<b>/proc/PID/exe</b> symlink to the process's binary magic: works even if the binary has been deleted!
	<b>/proc/PID/envIRON</b> all of the process's environment variables	<b>/proc/PID/status</b> Is the program running or asleep? How much memory is it using? And much more!
<b>/proc/PID/fd</b> Directory with every file the process has open! Run <code>\$ls -l /proc/42/fd</code> to see the list of files for process 42.  These symlinks are also magic & you can use them to recover deleted files♥	<b>/proc/PID/stack</b> The kernel's current stack for the process. Useful if it's stuck in a system call	and  Look at <b>man proc</b> for more information!
	<b>/proc/PID/maps</b> List of process's memory maps. Shared libraries, heap, anonymous maps, etc.	

Figura 6: Diretório /proc. Créditos: Julia Evans.

- Um **processo pai** cria **processos filhos**, que, por sua vez, podem criar outros processos, formando uma **árvore de processos**.
- Geralmente, os processos são identificados e gerenciados através de um **identificador de processo (PID)**.

- O processo pai e os filhos podem compartilhar recursos (mas não necessariamente).
- Opções de Execução:
  - ▶ O **pai e os filhos** executam **concorrentemente**.
  - ▶ O **pai espera** até que os **filhos terminem** sua execução.

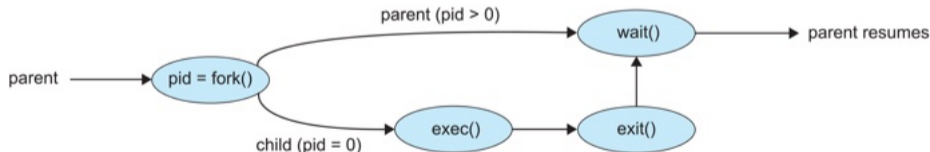


Figura 7: Criação de processo com o uso da chamada de sistema `fork()`.

Créditos: Silberschatz, Galvin and Gagne, 2018.

- O processo executa sua última instrução e solicita ao sistema operacional sua remoção usando a chamada de sistema `exit()`.
- O processo filho retorna um **status** para o processo pai através da chamada `wait()`.
- Os recursos utilizados pelo processo são desalocados pelo sistema operacional.
- O processo pai pode encerrar a execução de processos filhos usando a chamada de sistema `abort()`.
- Razões para encerramento de um processo filho:
  - ▶ O filho excedeu os recursos alocados.
  - ▶ A tarefa atribuída ao filho não é mais necessária.
  - ▶ O pai está finalizando e o sistema operacional não permite que o filho continue se o pai for encerrado.
- **Zumbi**: Um processo que foi encerrado, mas cujo pai ainda não chamou `wait()`.
- **Órfãos**: Um processo cujo pai não invocou `wait()` e, em vez disso, foi encerrado.

- O processo executa sua última instrução e solicita ao sistema operacional sua remoção usando a chamada de sistema `exit()`.
- O processo filho retorna um **status** para o processo pai através da chamada `wait()`.
- Os recursos utilizados pelo processo são desalocados pelo sistema operacional.
- O processo pai pode encerrar a execução de processos filhos usando a chamada de sistema `abort()`.
- Razões para encerramento de um processo filho:
  - ▶ O filho excedeu os recursos alocados.
  - ▶ A tarefa atribuída ao filho não é mais necessária.
  - ▶ O pai está finalizando e o sistema operacional não permite que o filho continue se o pai for encerrado.
- **Zumbi**: Um processo que foi encerrado, mas cujo pai ainda não chamou `wait()`.
- **Órfãos**: Um processo cujo pai não invocou `wait()` e, em vez disso, foi encerrado.



- O processo executa sua última instrução e solicita ao sistema operacional sua remoção usando a chamada de sistema `exit()`.
- O processo filho retorna um **status** para o processo pai através da chamada `wait()`.
- Os recursos utilizados pelo processo são desalocados pelo sistema operacional.
- O processo pai pode encerrar a execução de processos filhos usando a chamada de sistema `abort()`.
- Razões para encerramento de um processo filho:
  - ▶ O filho excedeu os recursos alocados.
  - ▶ A tarefa atribuída ao filho não é mais necessária.
  - ▶ O pai está finalizando e o sistema operacional não permite que o filho continue se o pai for encerrado.
- **Zumbi**: Um processo que foi encerrado, mas cujo pai ainda não chamou `wait()`.
- **Órfãos**: Um processo cujo pai não invocou `wait()` e, em vez disso, foi encerrado.

- O processo executa sua última instrução e solicita ao sistema operacional sua remoção usando a chamada de sistema `exit()`.
- O processo filho retorna um **status** para o processo pai através da chamada `wait()`.
- Os recursos utilizados pelo processo são desalocados pelo sistema operacional.
- O processo pai pode encerrar a execução de processos filhos usando a chamada de sistema `abort()`.
- Razões para encerramento de um processo filho:
  - O filho excedeu os recursos alocados.
  - A tarefa atribuída ao filho não é mais necessária.
  - O pai está finalizando e o sistema operacional não permite que o filho continue se o pai for encerrado.
- **Zumbi**: Um processo que foi encerrado, mas cujo pai ainda não chamou `wait()`.
- **Órfãos**: Um processo cujo pai não invocou `wait()` e, em vez disso, foi encerrado.

- O processo executa sua última instrução e solicita ao sistema operacional sua remoção usando a chamada de sistema `exit()`.
- O processo filho retorna um **status** para o processo pai através da chamada `wait()`.
- Os recursos utilizados pelo processo são desalocados pelo sistema operacional.
- O processo pai pode encerrar a execução de processos filhos usando a chamada de sistema `abort()`.
- Razões para encerramento de um processo filho:
  - ▶ O filho **excedeu** os recursos alocados.
  - ▶ A tarefa atribuída ao filho **não é mais necessária**.
  - ▶ O pai está finalizando e o sistema operacional **não permite que o filho continue** se o pai for encerrado.
- **Zumbi**: Um processo que foi encerrado, mas cujo pai ainda não chamou `wait()`.
- **Órfãos**: Um processo cujo pai não invocou `wait()` e, em vez disso, foi encerrado.

- O processo executa sua última instrução e solicita ao sistema operacional sua remoção usando a chamada de sistema `exit()`.
- O processo filho retorna um **status** para o processo pai através da chamada `wait()`.
- Os recursos utilizados pelo processo são desalocados pelo sistema operacional.
- O processo pai pode encerrar a execução de processos filhos usando a chamada de sistema `abort()`.
- Razões para encerramento de um processo filho:
  - ▶ O filho **excedeu** os recursos alocados.
  - ▶ A tarefa atribuída ao filho **não é mais necessária**.
  - ▶ O pai está finalizando e o sistema operacional **não permite que o filho continue** se o pai for encerrado.
- **Zumbi**: Um processo que foi encerrado, mas cujo pai ainda não chamou `wait()`.
- **Órfãos**: Um processo cujo pai não invocou `wait()` e, em vez disso, foi encerrado.

- O processo executa sua última instrução e solicita ao sistema operacional sua remoção usando a chamada de sistema `exit()`.
- O processo filho retorna um **status** para o processo pai através da chamada `wait()`.
- Os recursos utilizados pelo processo são desalocados pelo sistema operacional.
- O processo pai pode encerrar a execução de processos filhos usando a chamada de sistema `abort()`.
- Razões para encerramento de um processo filho:
  - ▶ O filho **excedeu** os recursos alocados.
  - ▶ A tarefa atribuída ao filho **não é mais necessária**.
  - ▶ O pai está finalizando e o sistema operacional **não permite que o filho continue** se o pai for encerrado.
- **Zumbi**: Um processo que foi encerrado, mas cujo pai ainda não chamou `wait()`.
- **Órfãos**: Um processo cujo pai não invocou `wait()` e, em vez disso, foi encerrado.

- O processo executa sua última instrução e solicita ao sistema operacional sua remoção usando a chamada de sistema `exit()`.
- O processo filho retorna um **status** para o processo pai através da chamada `wait()`.
- Os recursos utilizados pelo processo são desalocados pelo sistema operacional.
- O processo pai pode encerrar a execução de processos filhos usando a chamada de sistema `abort()`.
- Razões para encerramento de um processo filho:
  - ▶ O filho **excedeu** os recursos alocados.
  - ▶ A tarefa atribuída ao filho **não é mais necessária**.
  - ▶ O pai está finalizando e o sistema operacional **não permite que o filho continue** se o pai for encerrado.
- **Zumbi**: Um processo que foi encerrado, mas cujo pai ainda não chamou `wait()`.
- **Órfãos**: Um processo cujo pai não invocou `wait()` e, em vez disso, foi encerrado.

- O processo executa sua última instrução e solicita ao sistema operacional sua remoção usando a chamada de sistema `exit()`.
- O processo filho retorna um **status** para o processo pai através da chamada `wait()`.
- Os recursos utilizados pelo processo são desalocados pelo sistema operacional.
- O processo pai pode encerrar a execução de processos filhos usando a chamada de sistema `abort()`.
- Razões para encerramento de um processo filho:
  - ▶ O filho **excedeu** os recursos alocados.
  - ▶ A tarefa atribuída ao filho **não é mais necessária**.
  - ▶ O pai está finalizando e o sistema operacional **não permite que o filho continue** se o pai for encerrado.
- **Zumbi**: Um processo que foi encerrado, mas cujo pai ainda não chamou `wait()`.
- **Órfãos**: Um processo cujo pai não invocou `wait()` e, em vez disso, foi encerrado.

- O processo executa sua última instrução e solicita ao sistema operacional sua remoção usando a chamada de sistema `exit()`.
- O processo filho retorna um **status** para o processo pai através da chamada `wait()`.
- Os recursos utilizados pelo processo são desalocados pelo sistema operacional.
- O processo pai pode encerrar a execução de processos filhos usando a chamada de sistema `abort()`.
- Razões para encerramento de um processo filho:
  - ▶ O filho **excedeu** os recursos alocados.
  - ▶ A tarefa atribuída ao filho **não é mais necessária**.
  - ▶ O pai está finalizando e o sistema operacional **não permite que o filho continue** se o pai for encerrado.
- **Zumbi**: Um processo que foi encerrado, mas cujo pai ainda não chamou `wait()`.
- **Órfãos**: Um processo cujo pai não invocou `wait()` e, em vez disso, foi encerrado.



- Interrupções fazem com que o sistema operacional alterne a execução da CPU para uma rotina do kernel.
- Quando uma interrupção ocorre, o sistema deve salvar o **contexto do processo atual** para restaurá-lo depois.
- O contexto de um processo é armazenado no PCB.
- Alternar a CPU entre processos requer salvar o estado do processo atual e restaurar o estado do novo processo.
- Esse procedimento é chamado de **mudança de contexto**.
- O tempo de mudança de contexto é considerado **sobrecarga**, pois não realiza trabalho útil.

- Interrupções fazem com que o sistema operacional alterne a execução da CPU para uma rotina do kernel.
- Quando uma interrupção ocorre, o sistema deve salvar o **contexto do processo atual** para restaurá-lo depois.
- O contexto de um processo é armazenado no PCB.
- Alternar a CPU entre processos requer salvar o estado do processo atual e restaurar o estado do novo processo.
- Esse procedimento é chamado de **mudança de contexto**.
- O tempo de mudança de contexto é considerado **sobrecarga**, pois não realiza trabalho útil.

- Interrupções fazem com que o sistema operacional alterne a execução da CPU para uma rotina do kernel.
- Quando uma interrupção ocorre, o sistema deve salvar o **contexto do processo atual** para restaurá-lo depois.
- O contexto de um processo é armazenado no PCB.
- Alternar a CPU entre processos requer salvar o estado do processo atual e restaurar o estado do novo processo.
- Esse procedimento é chamado de **mudança de contexto**.
- O tempo de mudança de contexto é considerado **sobrecarga**, pois não realiza trabalho útil.

- Interrupções fazem com que o sistema operacional alterne a execução da CPU para uma rotina do kernel.
- Quando uma interrupção ocorre, o sistema deve salvar o **contexto do processo atual** para restaurá-lo depois.
- O contexto de um processo é armazenado no PCB.
- Alternar a CPU entre processos requer salvar o estado do processo atual e restaurar o estado do novo processo.
- Esse procedimento é chamado de **mudança de contexto**.
- O tempo de mudança de contexto é considerado **sobrecarga**, pois não realiza trabalho útil.

- Interrupções fazem com que o sistema operacional alterne a execução da CPU para uma rotina do kernel.
- Quando uma interrupção ocorre, o sistema deve salvar o **contexto do processo atual** para restaurá-lo depois.
- O contexto de um processo é armazenado no PCB.
- Alternar a CPU entre processos requer salvar o estado do processo atual e restaurar o estado do novo processo.
- Esse procedimento é chamado de **mudança de contexto**.
- O tempo de mudança de contexto é considerado **sobrecarga**, pois não realiza trabalho útil.

- Interrupções fazem com que o sistema operacional alterne a execução da CPU para uma rotina do kernel.
- Quando uma interrupção ocorre, o sistema deve salvar o **contexto do processo atual** para restaurá-lo depois.
- O contexto de um processo é armazenado no PCB.
- Alternar a CPU entre processos requer salvar o estado do processo atual e restaurar o estado do novo processo.
- Esse procedimento é chamado de **mudança de contexto**.
- O tempo de mudança de contexto é considerado **sobrecarga**, pois não realiza trabalho útil.

# Fechamento e Perspectivas

## ● **Resumo:**

- ▶ Sistemas Operacionais gerenciam a execução de múltiplos processos e threads simultaneamente.
- ▶ Multitarefa permite melhor **utilização dos recursos** da CPU.
- ▶ A **multitarefa** permite que vários processos compartilhem a CPU de forma eficiente.
- ▶ Um **processo** é uma instância em execução de um programa, enquanto uma **thread** é a menor unidade de execução dentro de um processo.

## ● **Principais Conceitos:**

- ▶ **Execução Concorrente vs. Paralela:** Concorrência alterna a execução entre threads, enquanto o paralelismo ocorre simultaneamente em múltiplos núcleos.
- ▶ **Threads vs. Processos:** Threads compartilham o mesmo espaço de memória do processo pai, enquanto processos são independentes entre si.

## ● **Próximos Passos:**

- ▶ Explorar informações sobre processos usando a linha de comando.
- ▶ Compreender o conceito de threads em detalhes.



# Dúvidas e Discussão

Prof. Dr. Denis M. L. Martins

[denis.mayr@puc-campinas.edu.br](mailto:denis.mayr@puc-campinas.edu.br)