

A close-up photograph of a computer processor (CPU) mounted on a printed circuit board (motherboard). The CPU is a light-colored, rectangular component with a metal heat spreader and several pins visible. The background is dark and out of focus, showing other components of the motherboard.

Sistemas Operacionais

Multitarefa: Processos e Threads

Pontifícia Universidade Católica de Campinas
Prof. Dr. Denis M. L. Martins

Objetivos de Aprendizado

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

Disclaimer

Parte do material apresentado a seguir foi adaptado de:

- [IT Systems – Open Educational Resource](#), produzido por [Jens~Lechtenböger](#); e
- [Open Education Hub - Operating Systems](#)

Imagens decorativas retiradas de [Unsplash](#)

Multitarefa

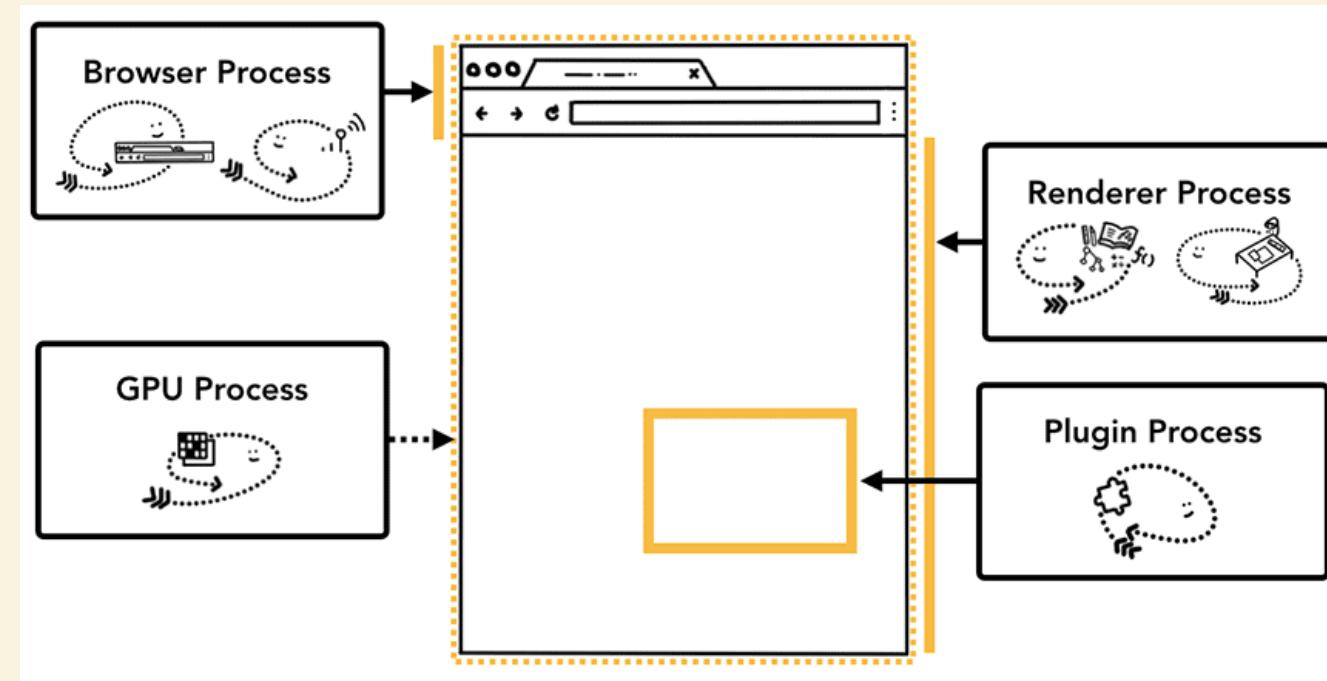
- Os sistemas operacionais permitem que múltiplas computações ocorram **concorrentemente** em um único sistema computacional.
 - Um exemplo de **multitarefa** ocorre quando você está **ouvindo música no Spotify enquanto navega na internet e clica em links**.
 - O **sistema operacional** gerencia a execução 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
- 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.

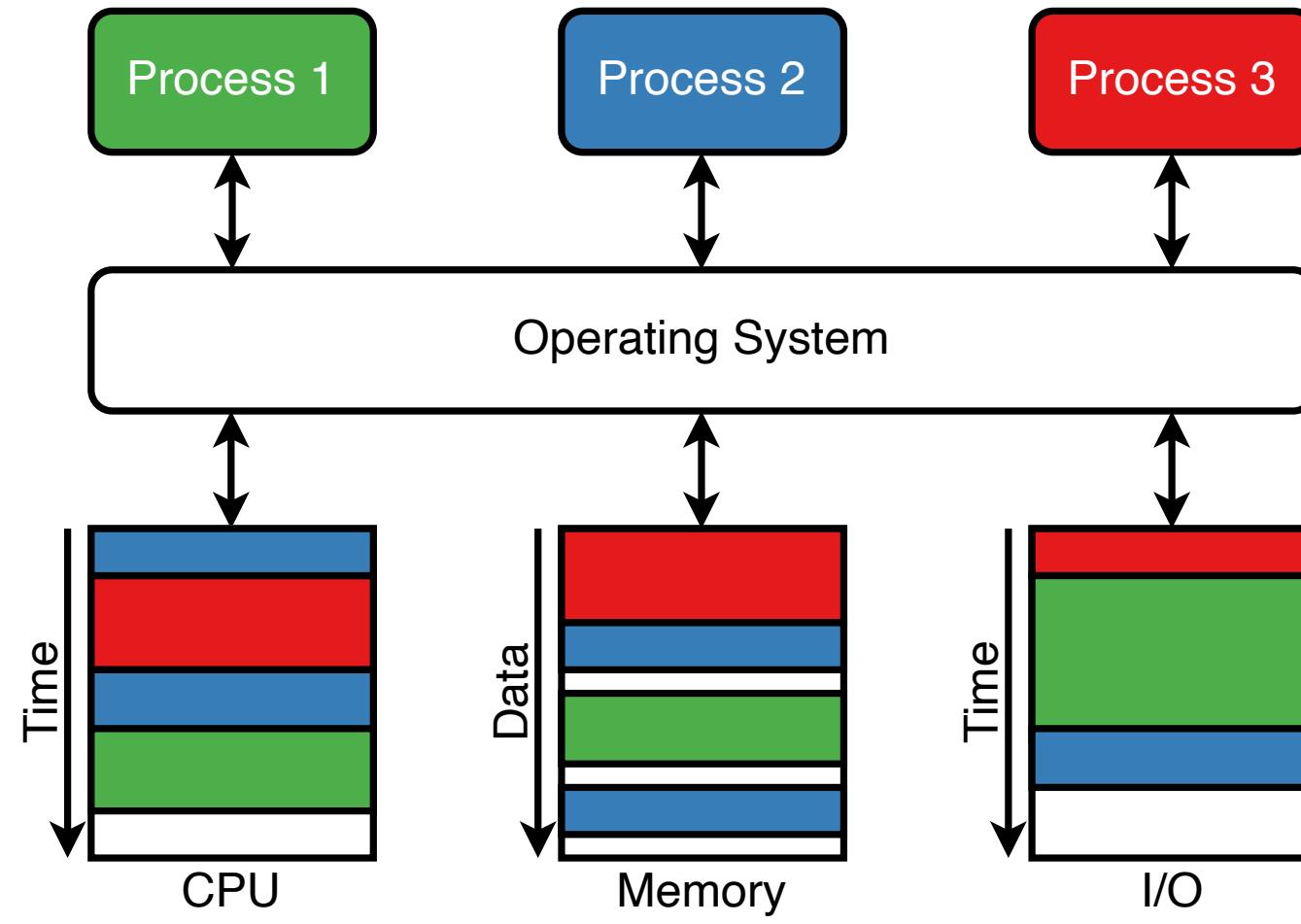
Processos em um SO

- Processo ≈ programa em execução.
 - Programa: entidade **passiva** guardada no disco (**arquivo executável**).
- Processo como unidade de **gerenciamento e proteção**.
 - Um programa pode criar múltiplos processos.



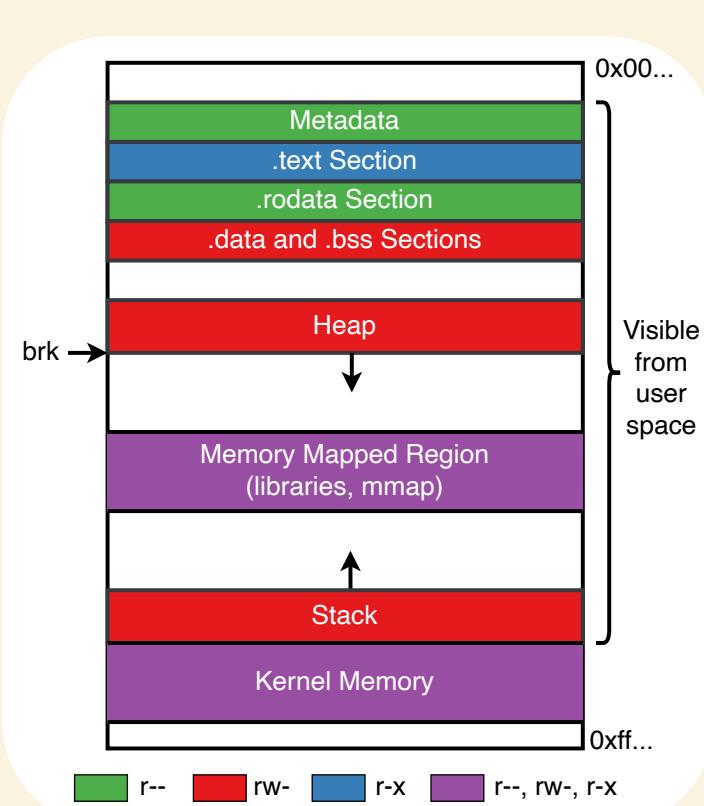
Na imagem: Processos apontando para diferentes partes da interface do usuário (UI) do navegador. Fonte: [Google Developers](#)

SO gerencia recursos para processos



Na Imagem: SO gerenciando memória, CPU e I/O para três processos diferentes. Fonte: [OER OS](#)

Bloco de Controle de Processo



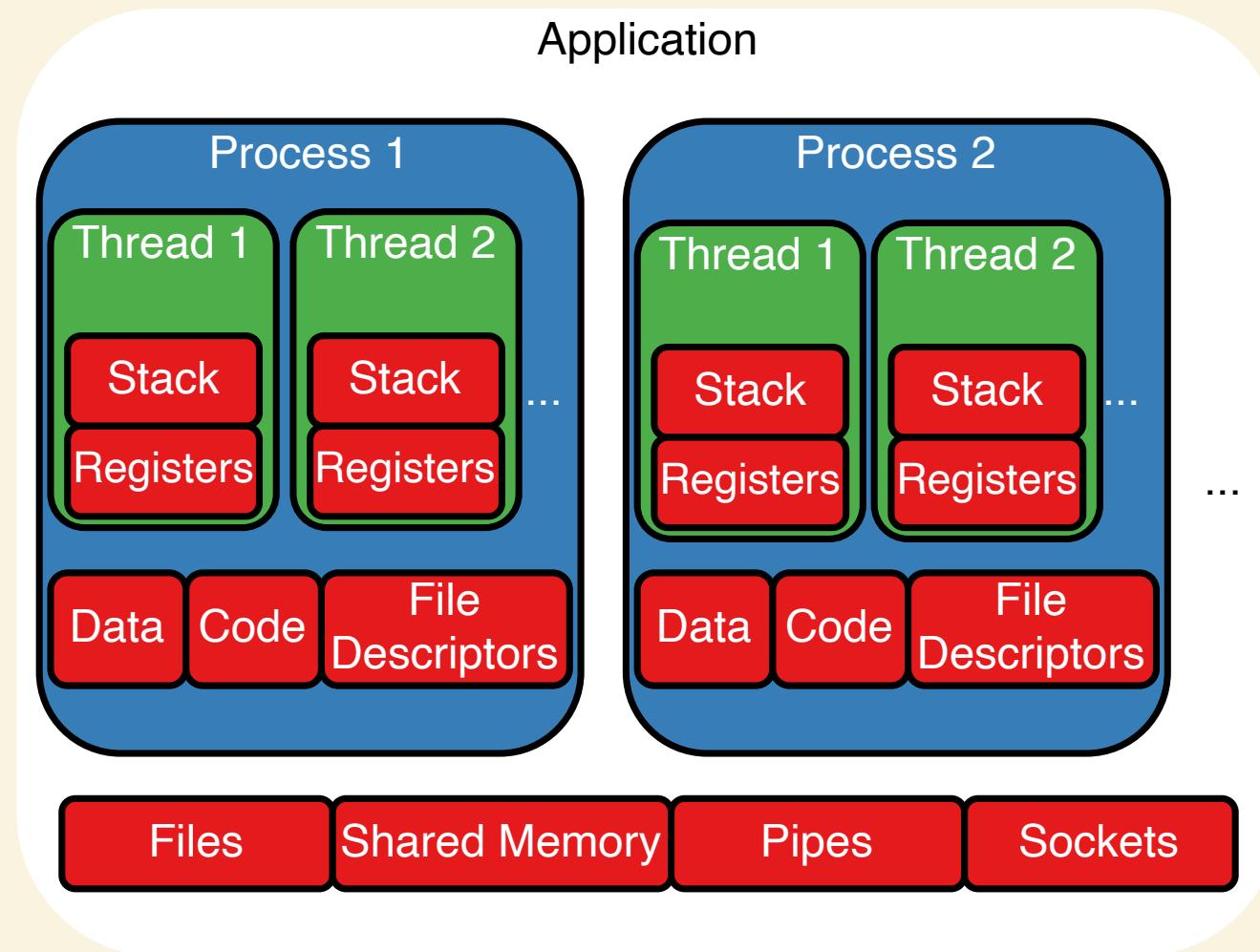
- **Estado do processo**: em execução, em espera, etc.
- **Contador de programa (PC)**: endereço da próxima instrução.
- **Registradores da CPU**: conteúdo de todos os registradores utilizados por processos.
- **Informação de gerenciamento de memória**: memória alocada para o processo.
- **Estatísticas**: uso de CPU, tempo desde o início, limites de tempo.
- **Informações de I/O**: dispositivos alocados ao processo, lista de arquivos abertos.
- Na imagem: layout de memória de um processo. Fonte: [OER OS](#).

Threads



- Thread: unidade de escalonamento do SO. Sequência independente de computações. São mais leves e mais fáceis de criar e destruir do que processos.
 - Um mesmo programa (processo) pode realizar várias tarefas concorrentemente.
 - Processador de texto (e.g. MS Word): processa texto do teclado, verifica ortografia, salva continuamente o documento → 3 threads num mesmo processo.
- Alto grau de independência: pense em funções diferentes no código.
- Núcleos individuais não estão se tornando significativamente mais rápidos. Relembre a [Lei de Moore](#). Threads permitem aproveitar o hardware atual.

Threads



Na Imagem: Aplicação com dois processos, cada processo com múltiplas threads. Fonte: [OER OS](#)

Criação de Thread

A thread é criada com `pthread_create` e esperada com `pthread_join`.

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

void *thread_function(void *arg) {
    printf("Olá do thread!\n");
    pthread_exit(0);
}

int main() {
    pthread_t thread1;
    pthread_create(&thread1, NULL, thread_function, NULL);
    pthread_join(thread1, NULL);
    return 0;
}
```

Múltiplas Threads

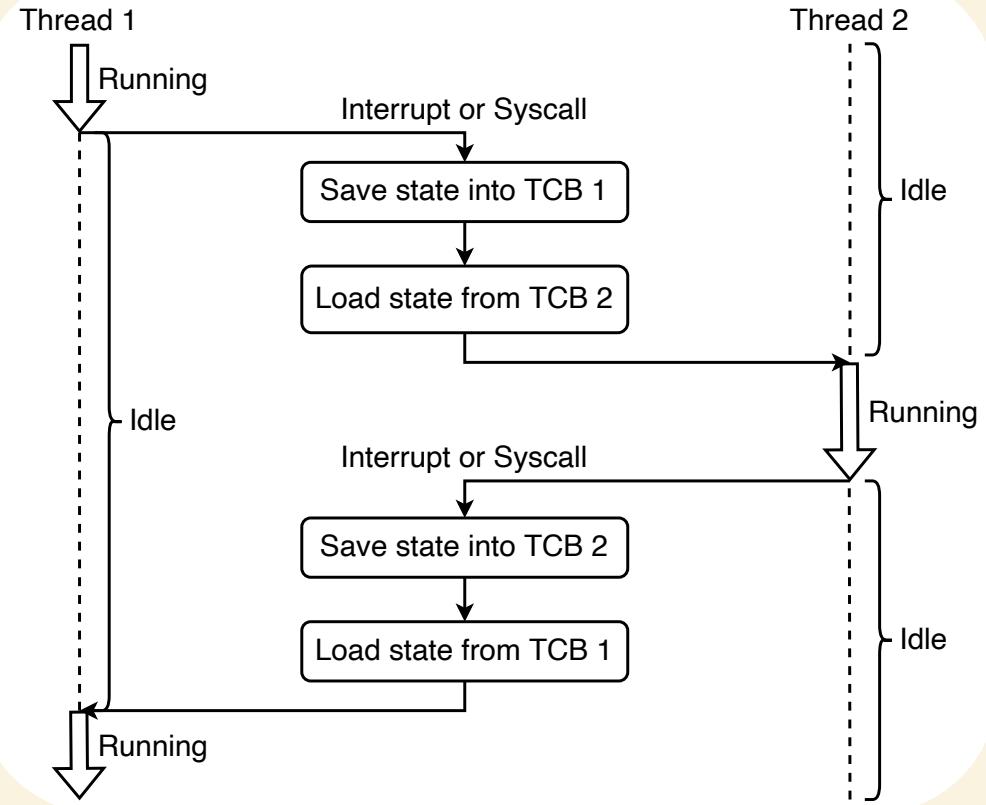
```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h> // Para sleep()

void *print_message(void *arg) {
    int thread_id = (int)(long)arg; /* Cast para obter o ID da thread */
    printf("Thread %d: Olá do thread!\n", thread_id);
    sleep(1); /* Pausa a execução da thread por 1 segundo */
    printf("Thread %d: Thread terminando...\n", thread_id);
    pthread_exit(0);
}

int main() {
    pthread_t threads[5]; /* Declara um array de threads (pthread_t) para armazenar os IDs das threads criadas */
    int thread_ids[5];

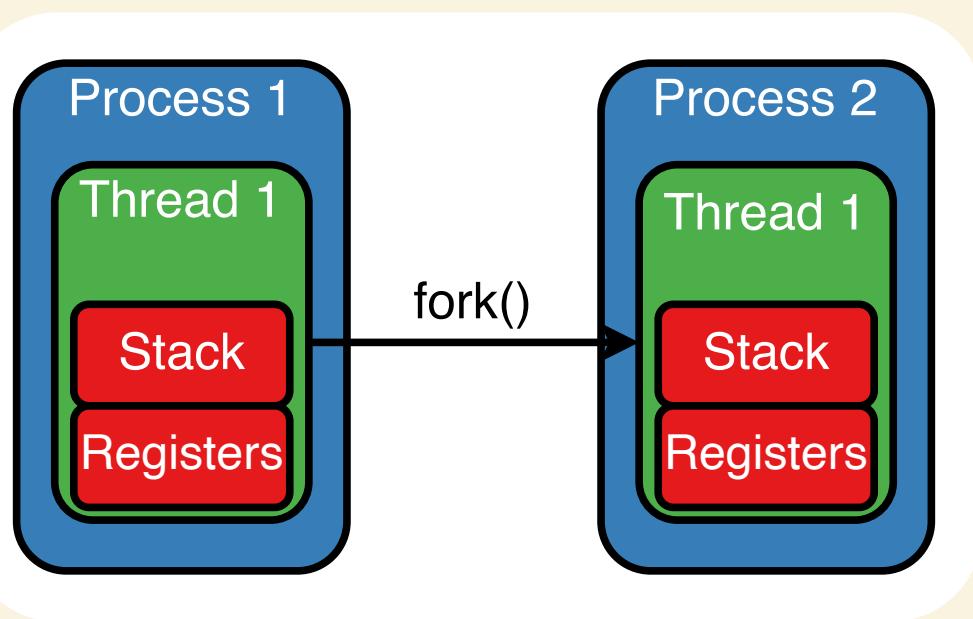
    /* Cria 5 threads */
    for (int i = 0; i < 5; i++) {
        thread_ids[i] = i + 1; /* Atribui um ID único para cada thread */
        pthread_create(&threads[i], NULL, print_message, (void *)thread_ids[i]);
    }
    /* Faz com que o programa principal espere até que as threads sejam finalizadas antes de continuar a execução */
    for (int i = 0; i < 5; i++) {
        pthread_join(threads[i], NULL);
    }
    printf("Programa principal terminado.\n");
    return 0;
}
```

Mudança de Contexto



- 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/thread atual para restaurá-lo depois.
- O contexto de um processo é armazenado no PCB/TCB.
- 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
- Na imagem: mudança de contexto durante a execução intercalada de duas threads. Fonte: [OER OS](#)

Criando Processos - `fork()`



- Processo-pai cria filhos, que podem criar outros processos, formando uma árvore de processos
- Um processo é identificado por um *process identifier* (pid)
- Usando `fork()`, o processo-filho é uma cópia do processo-pai.

```
int main(int argc, char *argv[]) {
    printf("Hello, world!\n");
    fork();
    printf("Goodbye!\n");
    return 0;
}
```

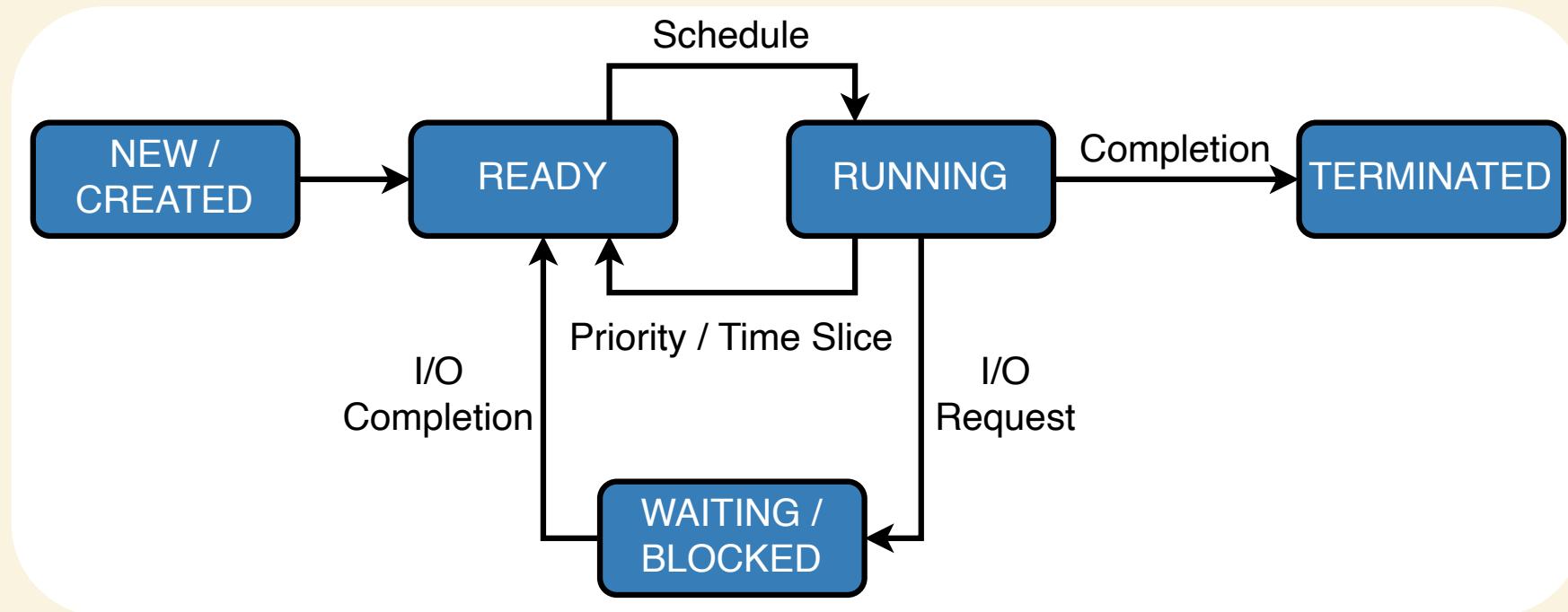
- Na imagem: Cópia do processo-pai pela função `fork()`. Fonte: [OER OS](#)

Criando processos - `fork()` (cont.)

```
//myprogram.c
#include <stdio.h>
#include <unistd.h>
int main(int argc, char *argv[]){
    pid_t pid = fork(); //Cria processo-filho
    if (pid < 0){
        printf("Erro ao criar processo.\n")
    }
    else if (pid == 0){
        printf("Eu sou o filho.\n");
        exit(1); //Encerra o processo
    }
    else {
        printf("Eu sou o pai.\n");
        wait(NULL); //Espera o processo-filho encerrar
    }
    return 0;
}
```

Diagrama de estados de threads/processos

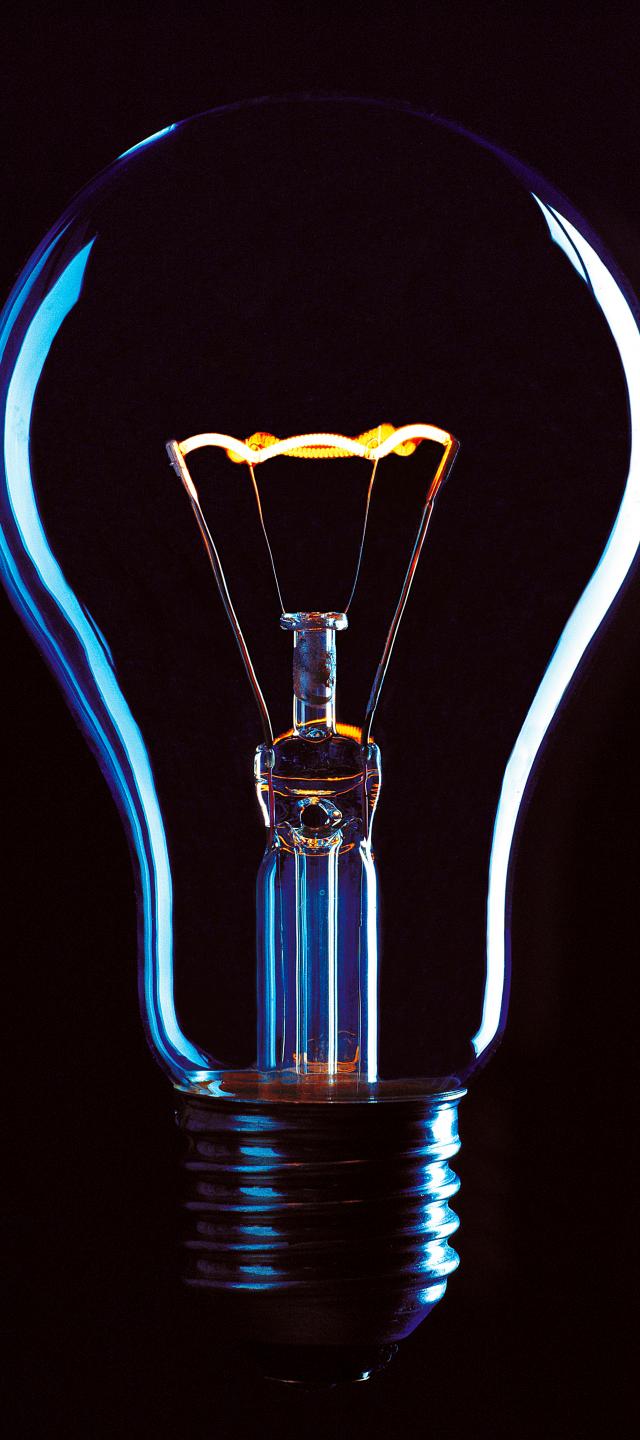
Conforme threads e processos executam, eles mudam de estado:



Fonte da Imagem: [OER OS](#)

Conclusão

- **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.
 - Melhora a **responsividade** do sistema.
 - Permite melhor **utilização dos recursos** da CPU.
- Um **processo** é uma instância em execução de um programa, enquanto uma **thread** é a menor unidade de execução dentro de um processo.
- **Execução Concorrente vs. Paralela:** Concorrência alterna a execução entre threads, enquanto o paralelismo ocorre simultaneamente em múltiplos núcleos.





Leitura Adicional

- Capítulo 2 do livro **Sistemas Operacionais Modernos**, de A. TANENBAUM
- Capítulo 3 do livro: **Operating Systems Concepts**, de A. SILBERCHATZ *et. al.*

Dúvidas e Discussão
