ALUNO: GUSTAVO HENRIQUE NOVAES GOMES
TURMA: A
PROFESSOR: GEORGE MENDES

# PROCESSOS EM SISTEMAS OPERACIONAIS
# PROCESSUAL - N1

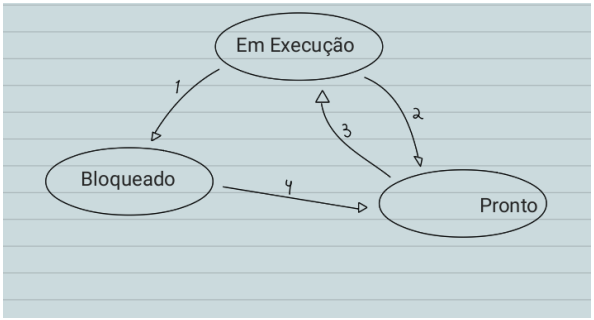
## 1. DIFERENÇA ENTRE PROGRAMA E PROCESSO:

É um conjunto de instruções escritas em alguma linguagem de programação. Fica armazenado no disco, esperando para ser executado. Tipo o arquivo .exe. de um jogo ou aplicativo.

Já o processo e o programa em execução. Quando você abre o programa, ele vira um processo. O sistema operacional cria um ambiente para ele funcionar: com memória, tempo de CPU etc.

## 2. ESTADOS DE UM PROCESSO E TRANSIÇÕES:

Os principais estados são: *Novo*, *Pronto*, *Executando*, *Bloqueado* e *Finalizado*. As transições ocorrem, por exemplo, quando o processo é admitido (Novo → Pronto), escalado pela CPU (Pronto → Executando), aguarda I/O (Executando → Bloqueado), ou termina sua execução (Executando → Finalizado).




## 3. CONTEÚDO DO PROCESS CONTROL BLOCK (PCB):

O PCB armazena informações essenciais do processo, como: identificador (PID), estado atual, registradores da CPU, ponteiros de pilha, contador de programa, informações de escalonamento, e recursos alocados.

## 4. Recursos ao término do processo:

Quando um processo termina, o sistema operacional precisa liberar tudo que ele estava usando. Isso inclui a memória que foi alocada, os arquivos que estavam abertos e os dispositivos que estavam em uso. Além disso, o sistema remove o bloco de controle do processo (PCB), que guardava informações sobre ele. Essa liberação é essencial para que os recursos fiquem disponíveis para outros processos e para manter o sistema funcionando de forma eficiente. Se isso não for feito corretamente, pode causar lentidão ou até travamentos.

## 5. Diferença entre fork() e exec() no UNIX:

No Unix, fork() cria um novo processo filho como cópia do processo pai. O pai e o filho continuam executando a partir do mesmo ponto, mas com contextos separados. Já exec() não cria processo, ele substitui o processo atual por um novo programa. Depois do exec(), o código antigo some e só o novo programa continua. Normalmente o shell usa fork() para gerar um filho e depois exec() dentro dele para rodar o comando desejado. Assim, o shell original não é perdido e continua esperando.

Resumo: fork() duplica, exec() troca.

## 6. Hierarquia de processos no UNIX:

No UNIX, processos são organizados em uma hierarquia pai-filho. O processo init (ou systemd) é o ancestral de todos os processos. Cada processo pode gerar filhos via fork(), formando uma árvore de processos.

## 7. Comparação entre memória compartilhada e troca de mensagens (IPC):

*Memória compartilhada* permite que processos acessem uma área comum de memória, sendo eficiente, mas exigindo sincronização. *Troca de mensagens* envolve envio/recebimento de dados via sistema operacional, sendo mais segura, porém menos eficiente.

## 8. Exemplos de chamadas de sistema em IPC:

Memória compartilhada: shmget(), shmat(), shmdt(), shmctl()

Troca de mensagens: msgget(), msgsnd(), msgrcv(), msgctl()

Pipes e sockets: pipe(), socket(), send(), recv()

## 9. Importância do gerenciamento de processos pelo SO:

O gerenciamento de processos garante que múltiplos processos coexistam de forma eficiente e segura, evitando conflitos por recursos, garantindo escalonamento justo e mantendo a estabilidade do sistema.

## 10. Diferença entre processos independentes e cooperativos:

Processos independentes não compartilham dados nem recursos com outros, ou seja, executam sem interferir ou depender de nenhum outro processo. Já processos cooperativos podem se comunicar, compartilhar informações e até sincronizar ações entre si, o que permite colaboração mas também traz riscos (como deadlock ou inconsistências se não houver controle).

## 11. Processo zumbi em UNIX/Linux:

Um *processo zumbi* é aquele que terminou sua execução, mas cujo PCB ainda está na tabela de processos aguardando que o processo pai leia seu status de término via wait(). Ele não consome recursos, mas ocupa espaço na tabela.

## 12. Diferença entre chamadas bloqueantes e não bloqueantes em IPC:

Chamadas bloqueantes suspendem o processo até que a operação seja concluída. Não bloqueantes retornam imediatamente, podendo indicar que a operação não foi realizada, exigindo verificação posterior.

## 13. Diferença entre processo pesado e thread:

Um processo possui seu próprio espaço de memória e recursos, enquanto uma thread compartilha o espaço de memória com outras threads do mesmo processo. Threads são mais leves e eficientes para tarefas concorrentes.

## 14. Troca de contexto em sistemas multiprogramados:

A troca de contexto permite que o sistema operacional salve o estado de um processo e carregue o estado de outro, possibilitando a execução concorrente e o compartilhamento da CPU entre múltiplos processos.

## 15. Vantagens e desvantagens da comunicação via memória compartilhada:

*Vantagens:* alta performance, baixo overhead, ideal para grandes volumes de dados.
*Desvantagens:* complexidade na sincronização, risco de condições de corrida, maior responsabilidade dos processos na integridade dos dados.