MÚLTIPLA ESCOLHA - ANALISTA DE SISTEMAS (Cesgranrio)

Um sistema operacional é formado por um conjunto de rotinas que oferecem serviços aos usuários, às suas aplicações e também ao próprio sistema. Esse conjunto de rotinas é denominado núcleo do sistema operacional ou kernel. A estrutura do sistema operacional, ou seja, a maneira como o código do sistema é organizado, pode variar conforme a concepção do projeto. A arquitetura monolítica pode ser definida como um programa:

a) composto por vários módulos que são compilados separadamente e depois linkados, formando um único programa executável.

b) composto por vários níveis sobrepostos, onde cada camada fornece um conjunto de funções que podem ser utilizadas apenas pelas camadas superiores.

c) composto por várias camadas, onde cada camada isola as funções do sistema operacional, facilitando sua manutenção e depuração, além de criar uma hierarquia de níveis de modos de acesso, protegendo as camadas mais internas.

d) formado por vários níveis, sendo que a camada de nível mais baixo é o hardware, e cada um dos níveis acima cria uma máquina virtual independente, em que cada uma oferece uma cópia virtual do hardware.

e) cujo núcleo do sistema é o menor e o mais simples possível e, neste caso, os servidores do sistema são responsáveis por oferecer um conjunto específico de funções, como gerência de arquivos, por exemplo.

MÚLTIPLA ESCOLHA - ANALISTA DE TI (UFPE)

O sistema operacional desempenha um papel importante no tratamento da E/S, atuando como interface entre o hardware e o software que solicita a E/S. Neste contexto é correto afirmar que:

a) os sistemas de E/S normalmente usam interrupções para comunicar informações sobre operações de E/S. Como essas interrupções causam uma transferência ao modo kernel ou supervisor, elas precisam ser tratadas pelo sistema operacional (SO).

b) não é responsabilidade do sistema operacional fornecer abstrações para acessar dispositivos nem fornecer rotinas que tratam as operações de baixo nível dos dispositivos.

c) o sistema operacional tenta oferecer acesso equilibrado aos recursos de E/S, mas não é responsabilidade do SO escalonar acessos a fim de melhorar a vazão do sistema.

d) o sistema operacional precisa ser capaz de dar comandos aos dispositivos E/S. Esses comandos incluem apenas operações como ler e escrever.

e) o sistema operacional precisa ser capaz de comunicar-se com os dispositivos de E/S mas não pode impedir que o programa do usuário se comunique com os dispositivos de E/S diretamente.

MÚLTIPLA ESCOLHA - ANALISTA DE SEGURANÇA (FGV)

Sistema Operacional é, por definição, um conjunto otimizado de programas que tem por objetivo gerenciar recursos dos computadores.
Nesse sentido, as funções de gerência desempenhadas pelos sistemas operacionais, incluem os seguintes componentes:

a) registradores, unidade de controle, unidade lógica e aritmética e barramentos de interconexão.
b) microprocessador, barramentos USB, slots de memória e controladoras de armazenamento.
c) floppy disk, disco rígido SATA, memória DDR e periféricos de input / output.
d) processamento, memória, dispositivos de entrada/saída e dados.
e) usuários, firewalls, equipamentos de segurança e software.

Processos

Sistemas Operacionais

Prof. Pedro Ramos
pramos.costar@gmail.com

Pontifícia Universidade Católica de Minas Gerais
ICEI - Departamento de Ciência da Computação

ANTERIORMENTE...

- Da arquitetura até o SO, do SO até o usuário:
  Recursos da arquitetura, gerenciamento do SO e abstrações de usuário:

HARDWARE	SERVIÇO DO SO	ABSTRAÇÃO DE USUÁRIO
Processador (CPU)	Gerenciamento de processos, escalonamento, interrupções, proteção, contabilidade, sincronização	Processo
Memória	Gerenciamento, proteção, memória virtual	Espaço de endereços
Dispositivos I/O (E/S)	Concorrência com CPU, tratamento de interrupções	Terminal, mouse, impressora, chamadas de sistema
Sistema de arquivos	Gerenciamento, persistência	Arquivos
Sistemas distribuídos	Redes, segurança	RPCs, sistemas de arquivo em nuvem

- Chamadas de sistema
- 4 arquiteturas estruturais para projetar Kernels

ANTERIORMENTE...

- Da arquitetura até o SO, do SO até o usuário:
  Recursos da arquitetura, gerenciamento do SO e abstrações de usuário:

HARDWARE	SERVIÇO DO SO	ABSTRAÇÃO DE USUÁRIO
Processador (CPU)	Gerenciamento de processos, escalonamento, interrupções, proteção, contabilidade, sincronização	Processo
Memória	Gerenciamento, proteção, memória virtual	Espaço de endereços
Dispositivos I/O (E/S)	Concorrência com CPU, tratamento de interrupções	Terminal, mouse, impressora, chamadas de sistema
Sistema de arquivos	Gerenciamento, persistência	Arquivos
Sistemas distribuídos	Redes, segurança	RPCs, sistemas de arquivo em nuvem

- Chamadas de sistema
- 4 arquiteturas estruturais para projetar Kernels

NESTA AULA: PROCESSOS

- O processo como unidade de execução
- Como processos são representados pelo SO?
- Quais são os possíveis estados de execução e como o sistema se move de um estado para outro?
- Como processos são criados no sistema?
- Como processos se comunicam? É eficiente?

NESTA AULA: PROCESSOS

- O processo como unidade de execução
- Como processos são representados pelo SO?
- Quais são os possíveis estados de execução e como o sistema se move de um estado para outro?
- Como processos são criados no sistema?
- Como processos se comunicam? É eficiente?

O QUE TEM DENTRO DE UM PROCESSO?

- QUAL A DIFERENÇA ENTRE UM PROCESSO E UM PROGRAMA?

O QUE TEM DENTRO DE UM PROCESSO?

- Um programa é apenas um conjunto de instruções e dados que a CPU pode rodar em qualquer instante de tempo.

- Um processo é uma unidade de código (e seus dados) **EM EXECUÇÃO**.

O programa é um conceito ESTÁTICO, o processo é um conceito DINÂMICO.

O QUE TEM DENTRO DE UM PROCESSO?

Definição: contexto dinâmico da execução de um programa em execução.

- Vários processos podem executar o mesmo programa, mas cada um é um processo distinto com seu próprio estado (exemplo: MS Word)

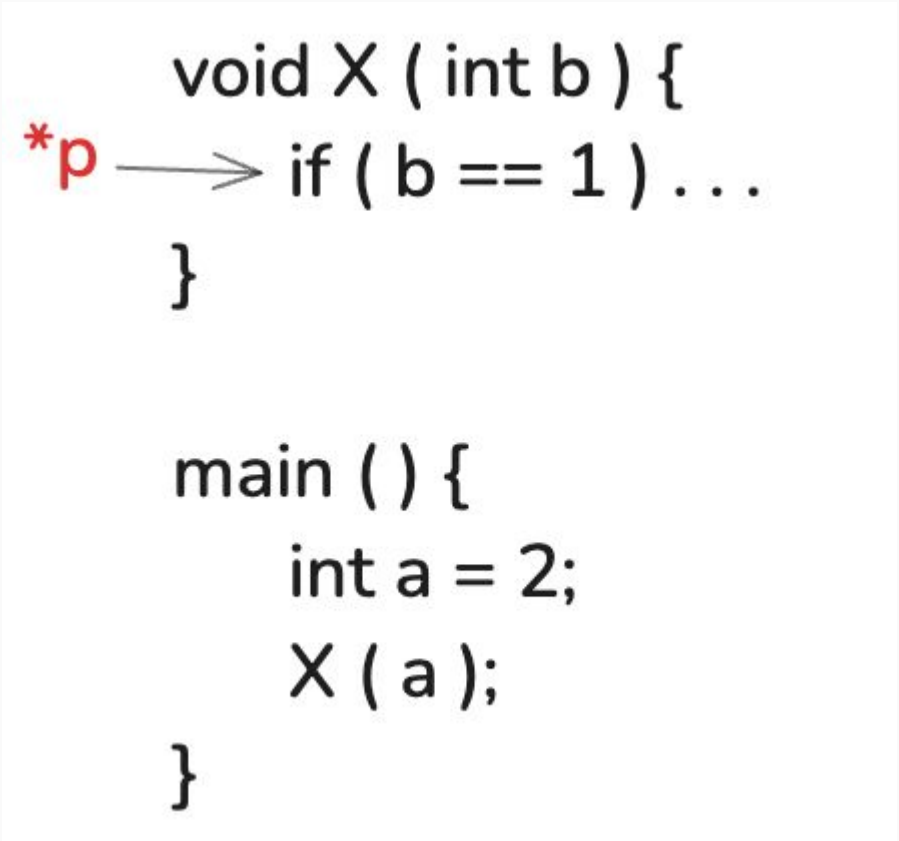
- Um processo executa SEQUENCIALMENTE, uma instrução por vez.

O QUE TEM DENTRO DE UM PROCESSO?

O ESTADO de um processo consiste em:

- O código do programa em execução;
- Os dados estáticos para executar o programa;
- Espaço para dados dinâmicos (heap), um apontador para o heap (HP)
- Apontador para a próxima instrução (PC)
- Uma pilha de execução com a cadeia de chamadas de função (pilha), o ponteiro para a pilha (SP)
- Valores dos registradores da CPU
- Conjunto de recursos em uso (arquivos abertos, por ex)
- Estado de execução: PRONTO | EXECUTANDO | ESPERANDO | etc

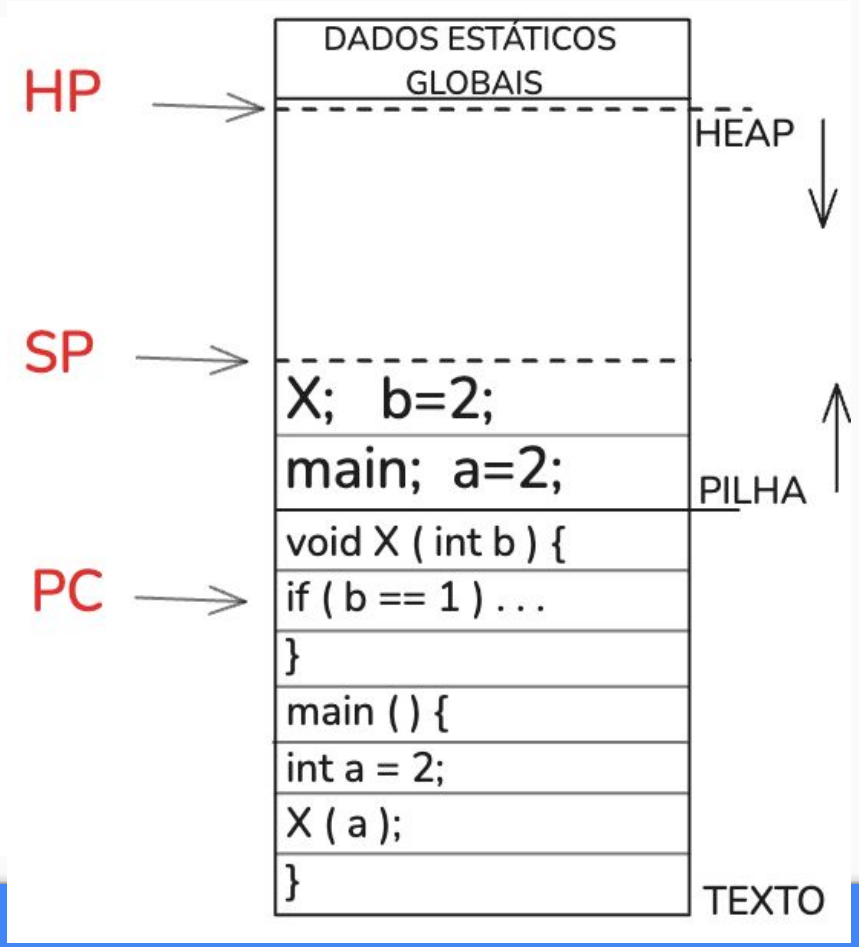
O QUE TEM DENTRO DE UM PROCESSO?



O QUE TEM DENTRO DE UM PROCESSO?

```
*p ──►  void X ( int b ) {
            if ( b == 1 ) . . .
        }

        main ( ) {
            int a = 2;
            X ( a );
        }
```

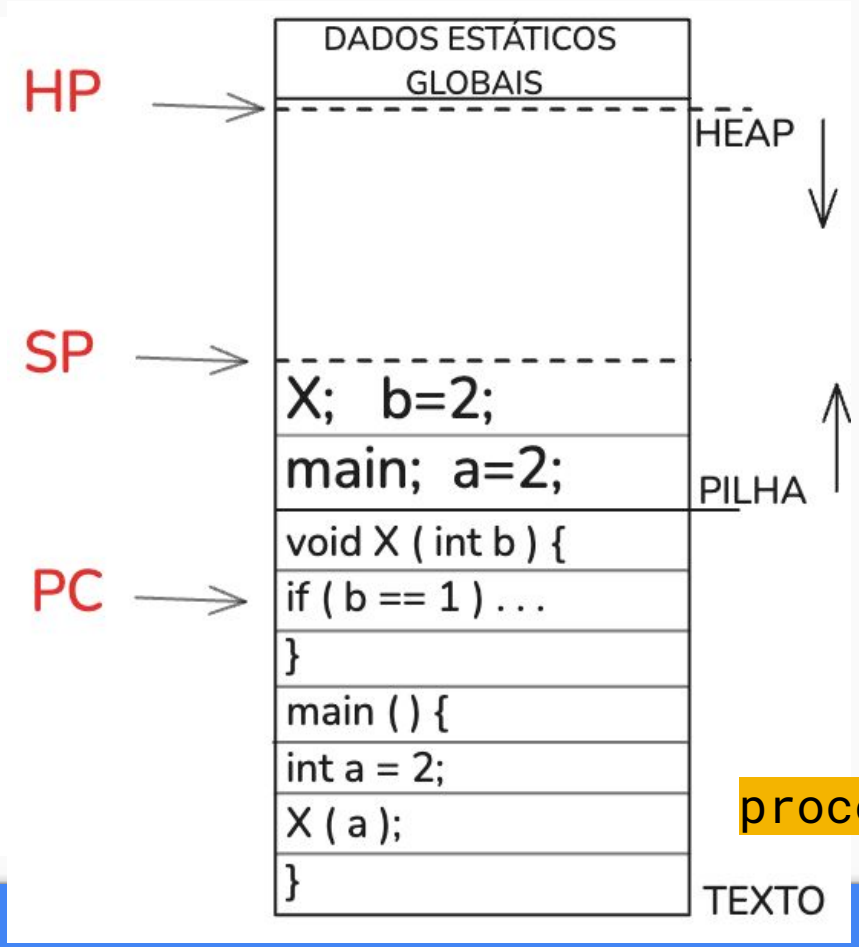


O QUE TEM DENTRO DE UM PROCESSO?

```
*p ──►  void X ( int b ) {
            if ( b == 1 ) . . .
        }

        main ( ) {
            int a = 2;
            X ( a );
        }
```

programa



processo

ESTADOS DO PROCESSO

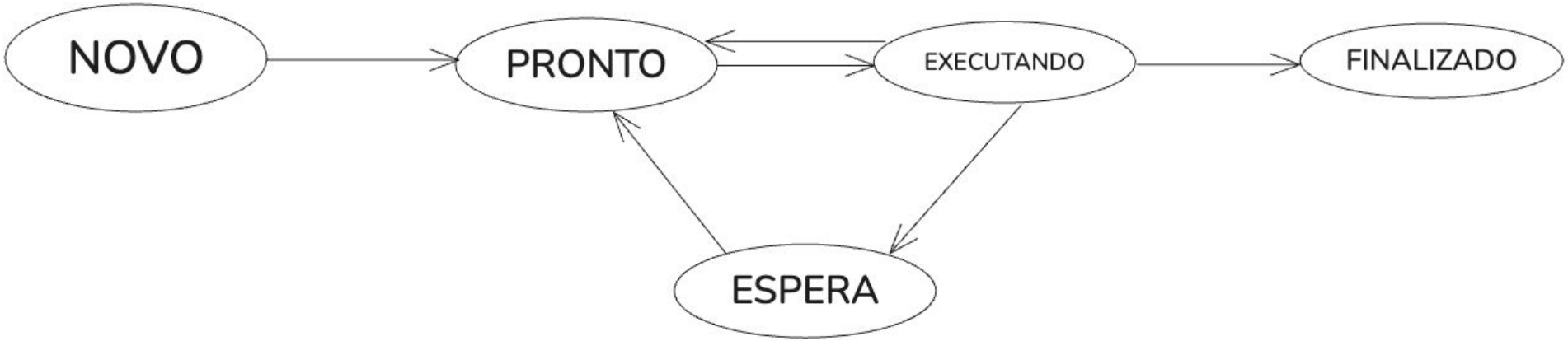
O ESTADO DE EXECUÇÃO indica o que o processo está fazendo:

- new (NOVO) → O SO está configurando o espaço para o processo
- running (EXECUTANDO) → O SO está executando as instruções do processo na CPU
- ready (PRONTO) → Está pronto para executar, mas esperando pela CPU
- waiting (EM ESPERA) → Está esperando por algum evento completar
- terminated (FINALIZADO) → O SO está destruindo o processo.

ESTADOS DO PROCESSO

O que faz o ESTADO do processo MUDAR enquanto o programa executa?

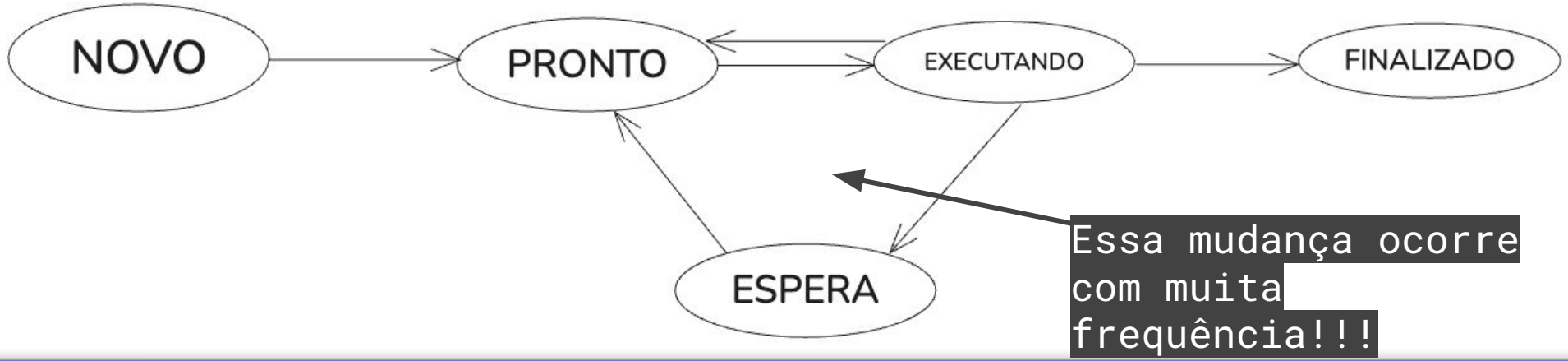
- AÇÕES DO PROGRAMA (chamadas de sistema),
- AÇÕES DO SO (escalonamento) e
- AÇÕES EXTERNAS (interrupções)



ESTADOS DO PROCESSO

O que faz o ESTADO do processo MUDAR enquanto o programa executa?

- AÇÕES DO PROGRAMA (chamadas de sistema),
- AÇÕES DO SO (escalonamento) e
- AÇÕES EXTERNAS (interrupções)

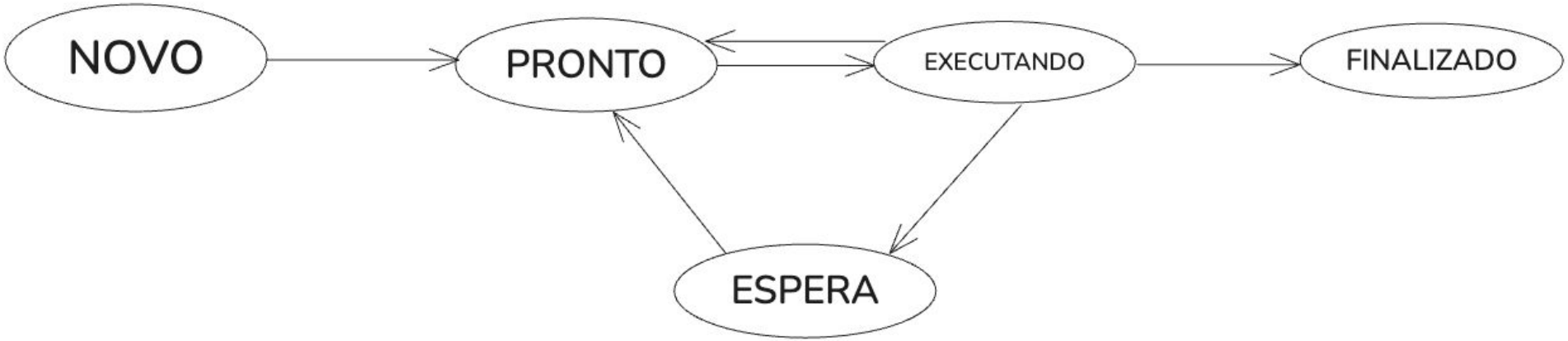


ESTADOS DO PROCESSO

Sequência de estados:

Exemplo:

```
void main() {
    printf(“Hello world”);
}
```



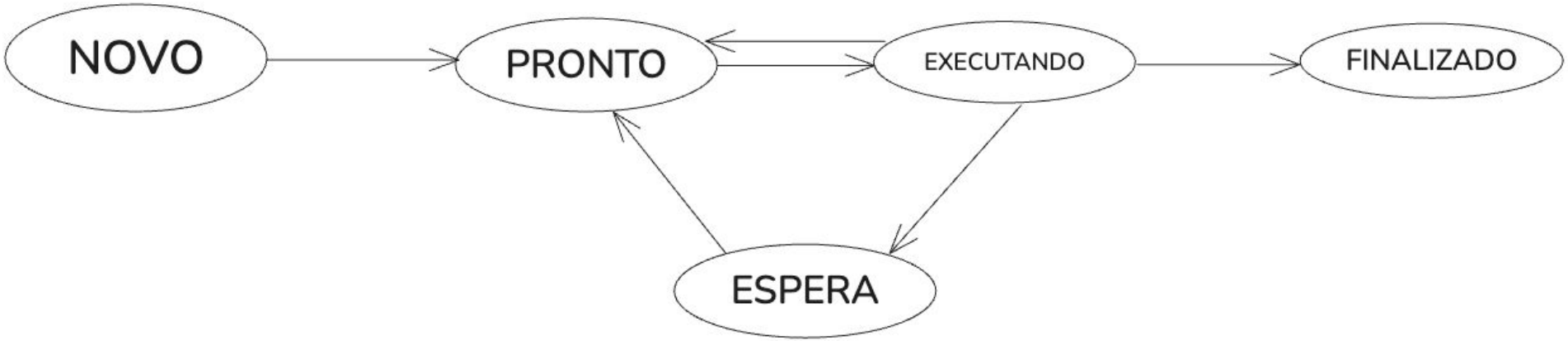
ESTADOS DO PROCESSO

Exemplo:

```
void main() {
    printf("Hello world");
}
```

Sequência de estados:

new	NOVO
ready	PRONTO
running	EXECUTANDO
waiting	ESPERANDO *(printf)*
ready	PRONTO
running	EXECUTANDO
terminated	FINALIZADO



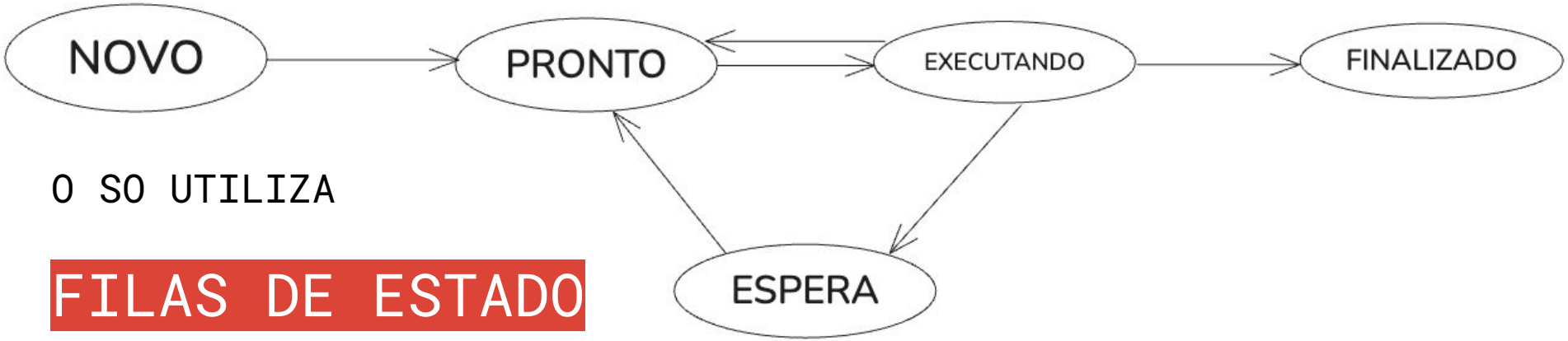
ESTADOS DO PROCESSO

Exemplo:

```
void main() {
    printf("Hello world");
}
```

Sequência de estados:

new	NOVO
ready	PRONTO
running	EXECUTANDO
waiting	ESPERANDO *(printf)*
ready	PRONTO
running	EXECUTANDO
terminated	FINALIZADO



O SO UTILIZA

FILAS DE ESTADO

A ESTRUTURA DE DADOS DO PROCESSO

- PCB - BLOCO DE CONTROLE DO PROCESSO (Process Control Block)
  - Rastreia o estado e local de cada processo
  - **O SO aloca** um novo PCB na **criação** de cada processo e coloca em uma fila de estados
  - **O SO desaloca** espaço do PCB quando o processo **termina**

A ESTRUTURA DE DADOS DO PROCESSO

- PCB - BLOCO DE CONTROLE DO PROCESSO (Process Control Block)
  - Rastreia o estado e local de cada processo
  - **O SO aloca** um novo PCB na **criação** de cada processo e coloca em uma fila de estados
  - **O SO desaloca** espaço do PCB quando o processo **termina**

O PCB contém:

Estado (espera, pronto, etc)
Número do processo
PC
SP
Registradores de propósito geral
Informação de memória virtual

username do dono
Lista de arquivos abertos
Ponteiros para filas de estado
Escalonamento info. (prioridade)
I/O status
...

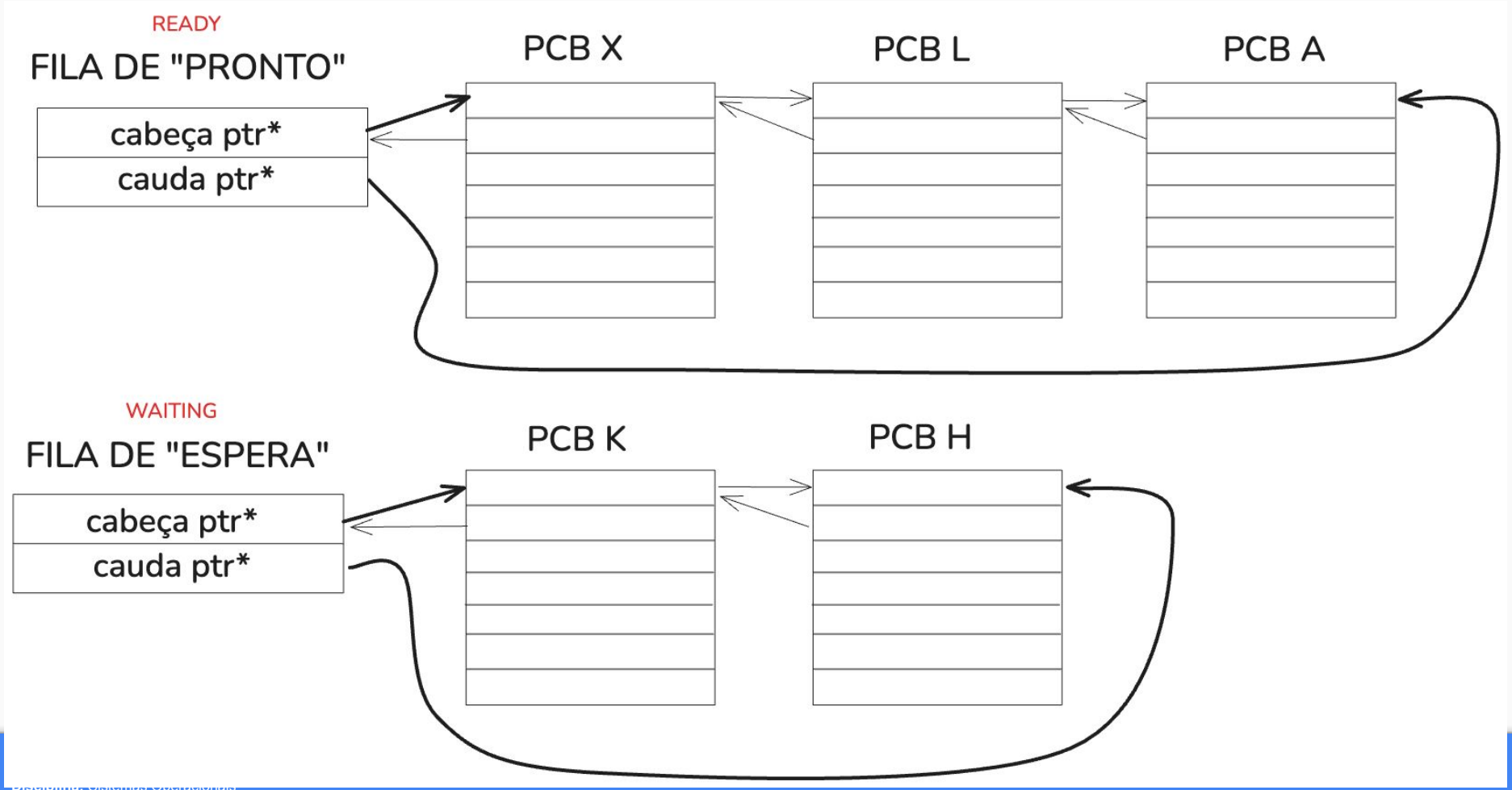
Exemplo - htop

Informação direto do PCB

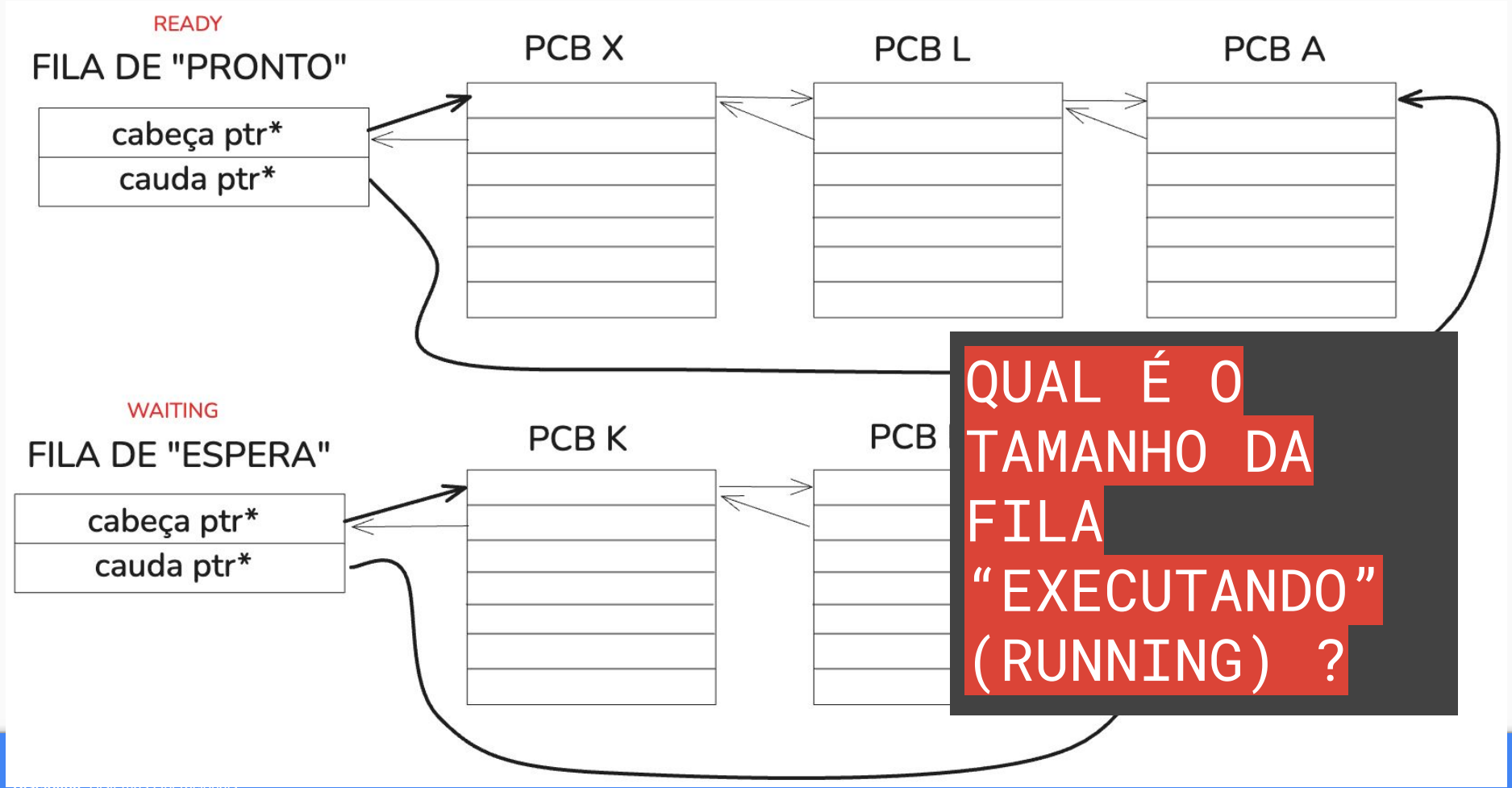
FILAS DE ESTADO

- O SO mantém todos os PCB's em FILAS de estados de acordo com o estado de execução do processo.
- Quando o SO muda o estado de um processo, o PCB também deve mudar de fila.
- O SO pode usar *DIFERENTES POLÍTICAS* para cada uma das filas.
- Cada dispositivo I/O tem sua própria fila de espera.

FILAS DE ESTADO



FILAS DE ESTADO



MUDANÇA DE CONTEXTO

- Interromper um processo e iniciar outro é chamado de mudança ou **TROCA DE CONTEXTO**

MUDANÇA DE CONTEXTO

- Interromper um processo e iniciar outro é chamado de mudança ou **TROCA DE CONTEXTO**

1. Carrega os valores do registradores especiais (PC, SP, etc) do PCB
2. Enquanto executa, a CPU muda os valores dos registradores
3. Quando interrompe, o SO salva os valores dos registradores no PCB (memória)

MUDANÇA DE CONTEXTO

- Interromper um processo e iniciar outro é chamado de mudança ou **TROCA DE CONTEXTO**

1. Carrega os valores do registradores especiais (PC, SP, etc) do PCB
2. Enquanto executa, a CPU muda os valores dos registradores
3. Quando interrompe, o SO salva os valores dos registradores no PCB (memória)

Este é o processo de TROCAR o CONTEXTO da CPU.
Sistemas atuais fazem em média 100 a 1000 mudanças de contexto por segundo. (2008: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/cc938606(v=technet.10)?redirectedfrom=MSDN)

PORQUÊ O CUSTO DA TROCA DE CONTEXTO É CARO?

CRIAÇÃO DE PROCESSO

- Um PROCESSO é criado somente a partir de OUTRO PROCESSO.

- O criador é chamado PROCESSO PAI (parent) e o novo de PROCESSO FILHO (child)
- O PROCESSO PAI define (ou doa) recursos e privilégios para seus filhos
- O pai pode ESPERAR o filho completar ou continuar em paralelo

- Em UNIX, o sistema utilizado é o *fork()*

CRIAÇÃO DE PROCESSO: *fork()*

- O comando fork() copia as variáveis e os registradores do PAI para o FILHO

CRIAÇÃO DE PROCESSO: *fork()*

- O comando fork() copia as variáveis e os registradores do PAI para o FILHO

Qual ESTRUTURA DE DADOS É UTILIZADA PARA REPRESENTAR ESSA SEMÂNTICA (pai e filho)?

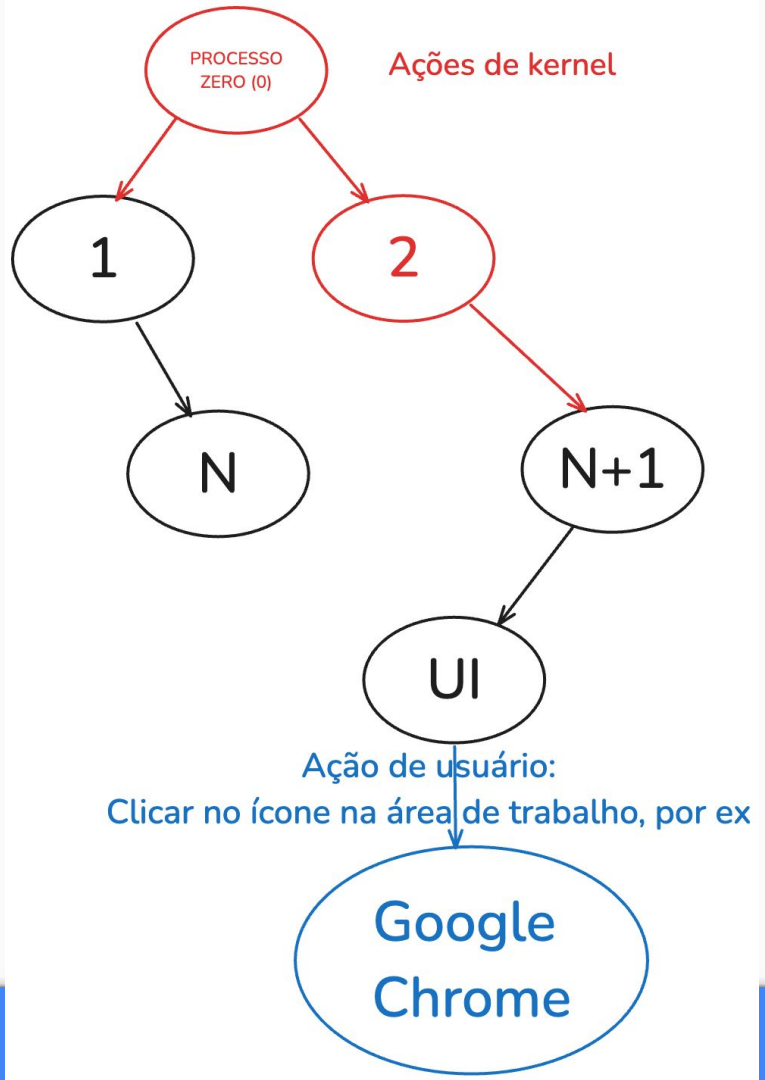
CRIAÇÃO DE PROCESSO: *fork()*

Fato:

UM PROCESSO SEMPRE É INICIADO A PARTIR DE OUTRO PROCESSO.

Pergunta: QUANDO EU ABRO O GOOGLE CHROME, QUAL PROCESSO CRIA O PROCESSO DO GOOGLE CHROME?

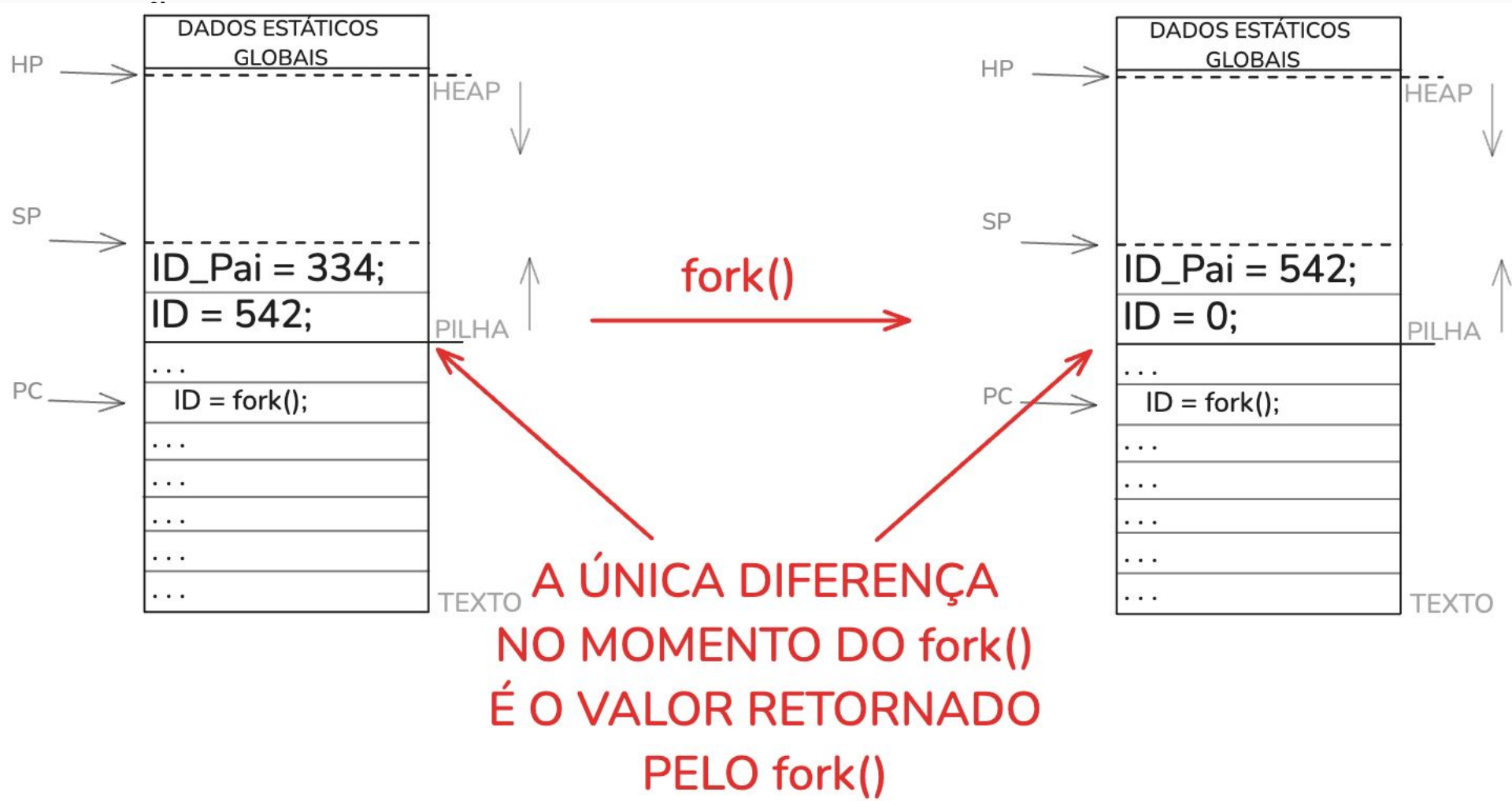
CRIAÇÃO DE PROCESSO: *fork()*



CRIAÇÃO DE PROCESSO: *fork()*

- O comando fork() copia as variáveis e os registradores do PAI para o FILHO

fork() realiza uma cópia idêntica do programa pai. A única diferença entre pai e filho e o **valor retornado** pelo fork
  - NO PAI, o fork vai retornar o ID do processo do filho
  - NO FILHO, o fork vai retornar 0


DADOS ESTÁTICOS GLOBAIS
HP
HEAP
SP
ID_Pai = 334;
ID = 542;
PILHA
PC
ID = fork();
TEXTO
fork()
DADOS ESTÁTICOS GLOBAIS
HP
HEAP
SP
ID_Pai = 542;
ID = 0;
PILHA
PC
ID = fork();
TEXTO
A ÚNICA DIFERENÇA
NO MOMENTO DO fork()
É O VALOR RETORNADO
PELO fork()

CRIAÇÃO DE PROCESSO: *fork()*

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

int main() {
    pid_t cid;  // Variável para guardar o PID

    printf("ANTES DO FORK, MEU ID É: %d, E O ID DO MEU PAI É: %d\n", getpid(), getppid());

    cid = fork();  // clona o processo

    if (cid == 0) {
        // Processo filho
        printf("EU SOU FILHO, MEU ID É: %d, E O ID DO MEU PAI É: %d\n", getpid(), getppid());

        // OPCIONAL: rodar exec() para substituir a imagem do programa
        // execlp("/bin/ls", "ls", NULL);

    } else if (cid > 0) {
        // Processo pai
        printf("DEPOIS DO FORK, MEU ID É: %d, E O ID DO MEU FILHO É: %d\n", getpid(), cid);
        sleep(1) // Dorme: evita o filho de se tornar um órfão antes do printf (do filho)

    } else {
        // fork() falhou
        printf("Falha na chamada de sistema!\n");
    }

    return 0;
}
```

CRIAÇÃO DE PROCESSO: *fork()*

- O comando fork() copia as variáveis e os registradores do PAI para o FILHO

fork() realiza uma cópia idêntica do programa pai.A única diferença entre pai e filho e o valor retornado pelo fork
    - NO PAI, o fork vai retornar o ID do processo do filho
    - NO FILHO, o fork vai retornar 0

- O pai pode esperar o filho terminar chamando a chamada de sistema *wait()* ou continuar execução

- O processo filho geralmente dispara um outro programa dentro de si mesmo com a chamada de sistema *exec()*

PROCESSOS ÓRFÃOS E PROCESSOS ZUMBI (zombie)

- PROCESSO ÓRFÃO: enquanto está executando, **não possui mais processo-pai vivo em memória.**

ID DO PAI ⇒ 1 (indica que não possui pai)

PROCESSOS ÓRFÃOS E PROCESSOS ZUMBI (zombie)

- PROCESSO ÓRFÃO: enquanto está executando, não possui mais processo-pai vivo em memória.

- PROCESSO ZUMBI:
  - - Processo pai cria o filho;
  - - Processo pai avisa para o filho que vai ler o valor de retorno (chamada *wait()*)
  - - Processo pai ainda não chamou *wait()*
  - - Filho espera que o pai espere por ele, ou seja, já terminou a execução mas **seu ID ainda consta na tabela de Processos do SO como um processo que ainda não terminou por completo**, pois o **pai não leu o valor de retorno.**

PROCESSOS ÓRFÃOS E PROCESSOS ZUMBI (zombie)

- PROCESSO ÓRFÃO: enquanto está executando, não possui mais processo-pai vivo em memória.

- PROCESSO ZUMBI:
  - SOLUÇÃO: Visitar a tabela de tempos em tempos para limpar os zumbis. Se o pai terminar, o filho se torna órfãos e o processo PID = 1 chama *wait()* para eles não se tornarem zumbis.

EXEC() -> SUBSTITUINDO A IMAGEM DO PROCESSO

- Basicamente **copia um novo código para dentro da região protegida de TEXTO** (segmento da memória).

EXEC() -> SUBSTITUINDO A IMAGEM DO PROCESSO

- Basicamente **copia um novo código para dentro da região protegida de TEXTO** (segmento da memória).

- exec(path): **Sobrescreve** a região TEXTO do programa em execução com o **conteúdo do caminho *path***

  Ou seja, substitui o texto do processo PAI por um novo texto.

É ISSO QUE O SHELL e a UI FAZEM QUANDO VOCÊ EXECUTA UM PROGRAMA.

EXERCÍCIO EM SALA (!)

```
...

1   int main() {
2       fork();
3       fork();
4       fork();
5
6       return 0;
7   }
```

1. **Desenhe a árvore de processos ao final da execução da linha 4.**
2. **Quantos processos foram executados no total, ao final da linha 6? Justifique sua resposta.**

ALGUMAS CHAMADAS DE SISTEMA

fork() -> bifurca um novo processo filho que é uma cópia do pai

execlp() -> substitui o programa do processo atual por outro

sleep() -> suspende execução de um processo por um período de tempo

waitpid() -> espera pelo processo PID terminar execução

gets() -> lê uma linha de um arquivo

COOPERAÇÃO ENTRE PROCESSOS

- Dois processos são INDEPENDENTES ou COOPERANTES.

- Processos cooperantes:
  - Atividade secundária em paralelo (melhorar performance)
  - Quebrar programas monolíticos
  - Compartilhar memória entre tarefas assíncronas

- É o ~~futuro~~ presente da computação.

COOPERAÇÃO ENTRE PROCESSOS: PRODUTORES E CONSUMIDORES

```
Produtor:

repetir pra sempre:
    produzir item;
    postar esse item no buffer;
```

- O produtor é a rotina ou thread MASTER/PRIMÁRIA

```
Consumidor:

repetir pra sempre:
    consumir item do buffer;
```

- Os consumidores são as rotinas SECUNDÁRIAS/SUBORDINADAS

ENVIO DE MENSAGENS

- O SO DEVE PROVER UM SISTEMA (modo kernel) PARA ENVIO DE MENSAGENS ENTRE PROCESSOS.

De modo geral:

```
main()
    ...
    if (fork() != 0) PRODUTOR()
    else CONSUMIDOR()
    ...
```

```
produtor:
    send(mensagem, consumidor)
```

```
consumidor:
    receive(mensagem, produtor)
```

ENVIO DE MENSAGENS

- Lembre-se: mensagens são referências para a memória.
  - O SO fornece um mecanismo para processos **se comunicarem**
    - O produtor escreve uma referência em uma região de memória Y
    - E o consumidor lê essa referência da região Y

- Pode ser feito de maneira *blocante* ou não *blocante*

- A partir daí, processos podem ler regiões maiores com *aritmética de ponteiros*
  - *p aponta para o bit 0, então *p + 1 aponta para o bit 8

ENVIO DE MENSAGENS

- Sistemas distribuídos se comunicam por mensagens.
- Cada processo precisa conhecer o nome dos outros processos.
- O consumidor tem buffer INFINITO.
- O SO gerencia as mensagens (logs, persistência, notificações)

ENVIO DE MENSAGENS: MEMÓRIA COMPARTILHADA GLOBAL

- Um mapeamento entre o espaço de endereços do processo para um OBJETO GLOBAL que é acessado por todos os outros processos.

- chamada *mmap()* faz isso.

- Por exemplo, processos filhos que precisam acessar estruturas definidas no pai.
  - *Nesse caso, não vale a pena usar produtor/consumidor, mas permitir que o filho acesse as regiões do pai diretamente.*

PERGUNTAS?

MÚLTIPLA ESCOLHA

Acerca de processos dentro de um sistema operacional, assinale a alternativa correta.

A O espaço de endereçamento de um processo contém a pilha de execução de um programa executável.
B O sistema operacional encerra definitivamente a execução de um processo quando este consome seu tempo de CPU permitido.
C Processos podem se dividir em subprocessos, mas não podem invocar a execução de novos processos.
D O contador de programa é um valor reservado na CPU que informa o tamanho da fila de processos.
E Processos maliciosos não detectados por programas antivírus são denominados daemons.

MÚLTIPLA ESCOLHA - (INPE)

No contexto da informática, considerando que um processo pode estar em um dos três estados possíveis (executando, pronto ou bloqueado), analise as afirmativas a seguir e assinale (V) para a verdadeira e (F) para a falsa.

( ) Um processo no estado “executando” está fisicamente utilizando a CPU (Unidade de Processamento Central).
( ) Um processo no estado “pronto” significa que o mesmo foi executado com sucesso.
( ) Um processo no estado “bloqueado” não pode mais ser executado.

As afirmativas são, respectivamente,

A F – V – V.
B F – V – F.
C V – F – V.
D V – F – F.
E V – V – F.

MÚLTIPLA ESCOLHA - (UERJ)

Em relação aos estados possíveis de um processo em um sistema operacional moderno, quando um determinado processo é interrompido por uma solicitação de disco, esse processo entrará no estado de:

A pronto
B terminado
C em espera
D em execução

REFERÊNCIAS

- **TANENBAUM, Andrew.** Sistemas operacionais modernos.
- **SILBERSCHATZ, Abraham et al.** Fundamentos de sistemas operacionais: princípios básicos.
- **MACHADO, Francis; MAIA, Luiz Paulo.** Arquitetura de Sistemas Operacionais.
- **CARISSIMI, Alexandre et al.** Sistemas operacionais.