EXERCÍCIOS TEÓRICOS - PROCESSOS

LUCAS FERNANDES LEMES

1. Qual a diferença entre programa e processo?

- **Programa:** É um arquivo passivo, um conjunto de instruções armazenado em disco (ex: `chrome.exe`).
- **Processo:** É um programa em execução. É uma entidade ativa que possui recursos alocados, como memória e tempo de CPU.

2. Quais são os estados de um processo e quando ocorrem as transições?

- **Novo (New):** O processo está sendo criado.
- **Pronto (Ready):** O processo está pronto para ser executado e aguarda a CPU.
- **Executando (Running):** As instruções do processo estão sendo executadas pela CPU.
- **Esperando (Waiting/Blocked):** O processo aguarda um evento, como uma operação de E/S (leitura de disco, entrada do usuário).
- **Terminado (Terminated):** O processo concluiu sua execução.

Transições:

- **Novo → Pronto:** O sistema operacional admite o processo.
- **Pronto → Executando:** O escalonador (scheduler) escolhe o processo para usar a CPU.
- **Executando → Pronto:** O tempo de CPU do processo acaba (preempção) ou ele é interrompido.
- **Executando → Esperando:** O processo solicita uma operação de E/S ou aguarda um recurso.
- **Esperando → Pronto:** O evento que o processo esperava ocorreu.
- **Executando → Terminado:** O processo finaliza sua execução.

3. O que contém um Process Control Block (PCB)?

O PCB (Bloco de Controle de Processo) é uma estrutura de dados que armazena todas as informações sobre um processo. Contém:

- Estado do processo (pronto, executando, etc.).
- Program Counter (aponta para a próxima instrução a ser executada).
- Registradores da CPU.
- Informações de escalonamento (prioridade).

- Informações de gerenciamento de memória (ponteiros para tabelas de página).
- Informações de E/S (dispositivos alocados, arquivos abertos).

4. O que acontece com os recursos de um processo quando ele termina?

Quando um processo termina, o sistema operacional **libera todos os recursos** que foram alocados a ele (memória, arquivos, dispositivos de E/S) para que possam ser utilizados por outros processos.

5. Qual a diferença entre fork() e exec() no UNIX?

- `fork()`**:** Cria um novo processo (processo filho) que é uma **cópia exata** do processo que o chamou (processo pai).
- `exec()`**: Substitui** o programa em execução no processo atual por um novo programa. Ele não cria um novo processo.

6. Como funciona a hierarquia de processos em UNIX?

Em UNIX, os processos são organizados em uma árvore. Todo processo (exceto o primeiro, `init`) tem um processo pai. Quando um processo cria outro usando `fork()`, ele se torna o pai do novo processo, estabelecendo uma relação hierárquica.

7. Compare memória compartilhada e troca de mensagens (IPC).

Ambos são métodos de Comunicação entre Processos (IPC).

- **Memória Compartilhada:** Mais **rápida**. Processos leem e escrevem em uma mesma área de memória. A sincronização deve ser controlada pelo programador para evitar conflitos.
- **Troca de Mensagens:** Mais **lenta**, pois envolve o kernel para enviar e receber mensagens. É mais segura e simples de implementar, pois o sistema operacional gerencia a comunicação.

8. Cite exemplos de chamadas de sistema usadas em IPC.

- **Pipes:** `pipe()`, `read()`, `write()`
- **Memória Compartilhada:** `shmget()`, `shmat()`, `shmdt()`
- **Troca de Mensagens:** `msgget()`, `msgsnd()`, `msgrcv()`
- **Sockets (para comunicação em rede):** `socket()`, `bind()`, `connect()`, `send()`, `recv()`

9. Por que é importante que o sistema operacional faça gerenciamento de processos?

Para garantir que a CPU e outros recursos do sistema sejam utilizados de forma **eficiente, justa e segura**. O gerenciamento permite a concorrência (múltiplos processos rodando "ao mesmo tempo"), protege os processos uns dos outros e aloca os recursos necessários para a execução de cada um.

10. Explique a diferença entre processos independentes e processos cooperativos.

- **Processos Independentes:** Não compartilham dados e não afetam nem são afetados por outros processos.
- **Processos Cooperativos:** Podem afetar ou ser afetados por outros processos. Geralmente compartilham dados (via IPC) para trabalhar em uma tarefa comum.

11. O que é um processo zumbi em UNIX/Linux?

É um processo que **já terminou** sua execução, mas sua entrada ainda existe na tabela de processos do sistema. Isso ocorre porque o processo pai ainda não leu o status de término do filho. Processos zumbis não consomem CPU, apenas um mínimo de memória no sistema.

12. Explique a diferença entre chamadas bloqueantes e não bloqueantes em IPC.

- **Bloqueante (Síncrona):** A chamada suspende (bloqueia) a execução do processo até que a operação de comunicação seja concluída (ex: `recv()` que espera até uma mensagem chegar).
- **Não Bloqueante (Assíncrona):** A chamada retorna imediatamente, tenha a operação sido concluída ou não. O processo pode continuar executando e verificar o status da operação mais tarde.

13. Qual a diferença entre processo pesado (process) e thread (processo leve)?

- **Processo:** Possui seu próprio espaço de endereçamento de memória. Criar um processo é mais "caro" (lento e consome mais recursos).
- **Thread:** É uma unidade de execução dentro de um processo. Várias threads de um mesmo processo **compartilham o mesmo espaço de memória**, tornando a criação e a comunicação entre elas muito mais rápida e eficiente.

14. Por que sistemas operacionais multiprogramados precisam de troca de contexto (context switch)?

A troca de contexto é o mecanismo que permite ao sistema operacional **alternar a execução da CPU entre diferentes processos**. O SO salva o estado do processo atual (seu contexto, no PCB) e carrega o estado de um novo processo. Isso é fundamental para criar a ilusão de que múltiplos programas estão rodando simultaneamente.

15. Cite vantagens e desvantagens da comunicação via memória compartilhada.

- **Vantagens:**
    - **Velocidade:** É o método de IPC mais rápido, pois não há intervenção do kernel para a transferência de dados.
    - **Eficiência:** Ideal para transferir grandes volumes de dados.
- **Desvantagens:**
    - **Complexidade:** O programador é responsável por implementar mecanismos de sincronização (ex: semáforos) para evitar condições de corrida (race conditions).
    - **Segurança:** Menos seguro, pois um erro em um processo pode corromper a memória compartilhada e afetar outros.