1. Qual a diferença entre programa e processo?

Um programa é uma entidade estática, essencialmente um arquivo executável contendo um conjunto de instruções e dados armazenados em disco. Ele é passivo. Um processo, por outro lado, é a manifestação dinâmica de um programa em execução. Ele é uma entidade ativa que possui recursos alocados pelo sistema operacional, como um espaço de endereçamento de memória, tempo de CPU e um contador de programa que aponta para a próxima instrução a ser executada. Em suma, o programa é o código, o processo é o código em ação.

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

Um processo navega por um ciclo de vida definido por estados. Ele nasce como "Novo", entra na fila de "Pronto" aguardando sua vez na CPU, passa para "Executando" quando o escalonador o seleciona, e pode ser movido para "Esperando" se precisar de um evento externo, como uma operação de entrada e saída. As transições são cruciais: o despachante move um processo de Pronto para Executando; uma interrupção por preempção (como o fim da sua fatia de tempo) o devolve para Pronto; uma solicitação de E/S o leva para Esperando; e a conclusão do evento o retorna para a fila de Pronto, reiniciando o ciclo até sua conclusão, quando entra no estado "Terminado".

3. O que contém um Bloco de Controle de Processo (BCP)?

O Bloco de Controle de Processo é a estrutura de dados fundamental que representa um processo para o sistema operacional. Pense nele como o "cérebro" administrativo do processo, contendo toda a informação de contexto necessária para o seu gerenciamento. Isso inclui o estado atual do processo, o valor do contador de programa, os conteúdos dos registradores da CPU, informações sobre o gerenciamento de memória (como ponteiros para tabelas de página), detalhes de escalonamento (prioridade) e informações sobre recursos alocados, como arquivos abertos e dispositivos.

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

Quando um processo termina sua execução, o sistema operacional inicia um procedimento de "limpeza" para reclamar todos os recursos que lhe foram alocados. Essa desalocação é fundamental para a saúde do sistema. O espaço de endereçamento de memória é liberado, todos os descritores de arquivos associados a arquivos abertos são fechados, conexões de rede são terminadas, e quaisquer outros recursos do núcleo, como semáforos ou segmentos de memória compartilhada, são devolvidos ao sistema para que possam ser reutilizados por outros processos.

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

As chamadas fork() e exec() são o mecanismo padrão no UNIX para criação de processos, mas atuam de forma distinta. A fork() é uma chamada de sistema que cria um novo processo-filho, duplicando o espaço de endereçamento do processo-pai. Já a família de chamadas exec() substitui completamente a imagem do processo atual por um novo programa. O padrão de uso clássico é um processo chamar fork() para se clonar e, em seguida, o processo-filho recém-criado chamar exec() para carregar e executar um programa diferente.

6. Como funciona a hierarquia de processos em UNIX?

Em sistemas do tipo UNIX, os processos são organizados em uma estrutura de árvore hierárquica. Todo processo, com exceção do processo inicial (o init ou systemd, com identificador 1), é criado por um processo pai, estabelecendo uma relação de parentesco. Isso cria uma genealogia clara, onde cada processo tem um único pai. Essa estrutura não é apenas organizacional; ela permite funcionalidades como o envio de sinais em cascata e o gerenciamento de grupos de processos.

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

Ambos são mecanismos de Comunicação entre Processos (IPC), mas com filosofias distintas. A memória compartilhada permite que múltiplos processos mapeiem uma mesma região da memória física em seus espaços de endereçamento. A comunicação ocorre de forma extremamente rápida, pois os dados não precisam ser copiados pelo núcleo do sistema. Em contrapartida, a troca de mensagens é mediada pelo núcleo: um processo envia uma mensagem através de uma chamada de sistema, o núcleo a armazena e o outro processo a lê com outra chamada. É mais lento devido à sobrecarga do núcleo, mas oferece sincronização implícita e maior segurança.

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

Para implementar a Comunicação entre Processos (IPC), utilizamos diferentes tipos de chamadas de sistema. No paradigma de troca de mensagens, existem chamadas para criar canais de comunicação (conhecidos como pipes) ou para estabelecer conexões mais flexíveis (os sockets). Já para memória compartilhada, o sistema oferece chamadas específicas para criar um segmento de memória, para anexá-lo ao espaço de endereçamento de um processo e, posteriormente, para desanexá-lo.

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

O gerenciamento de processos é a pedra angular dos sistemas operacionais multitarefa. Sua importância reside em prover abstração, proteção e compartilhamento de recursos. Ele abstrai a complexidade do hardware, oferecendo o modelo de processo. Garante a proteção isolando os espaços de memória, impedindo que um processo falho corrompa outros. E, o mais importante, gerencia o compartilhamento justo e eficiente dos recursos (principalmente a CPU e a memória) entre múltiplos processos concorrentes, maximizando a utilização do sistema e garantindo a responsividade.

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

Um processo independente é aquele cuja execução não é afetada por outros processos, e ele próprio não afeta nenhum outro. Ele opera em isolamento, sem compartilhar dados. Já um processo cooperativo pode afetar ou ser afetado por outros processos, tipicamente porque eles compartilham dados ou estado. A cooperação é essencial para tarefas que exigem compartilhamento de informações, modularidade ou paralelismo para ganho de desempenho, mas introduz a complexidade do controle de concorrência e da sincronização.

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

Um processo zumbi é um processo que já completou sua execução, mas sua entrada ainda persiste na tabela de processos do núcleo. Isso ocorre porque o processo pai ainda não coletou o status de terminação do filho através da chamada de sistema wait(). O processo em si não consome CPU nem memória, mas seu identificador (PID) e a estrutura no núcleo permanecem, consumindo um recurso finito. Um grande número de zumbis pode, eventualmente, esgotar os PIDs disponíveis, impedindo a criação de novos processos.

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

A distinção reside no comportamento do processo chamador. Uma chamada bloqueante (síncrona) suspende a execução do processo até que a operação de comunicação seja concluída. Por exemplo, uma leitura de um canal vazio fará o processo bloquear até que dados sejam escritos nele. Uma chamada não bloqueante (assíncrona) retorna imediatamente. Se a operação não pôde ser completada (o canal estava vazio), a chamada retorna um código de erro, permitindo que o processo execute outras tarefas em vez de ficar ocioso.

13. Qual a diferença entre processo e thread?

Um processo tradicional é uma unidade de alocação de recursos; ele possui seu próprio espaço de endereçamento de memória, descritores de arquivos, etc. Uma thread (ou fio de execução) é uma unidade de execução que opera dentro do contexto de um processo. Múltiplas threads de um mesmo processo compartilham o mesmo espaço de endereçamento e os mesmos recursos, mas cada uma tem sua própria pilha de execução e seu próprio contador de programa. Isso torna a criação de threads e a troca de contexto entre elas muito mais rápida do que entre processos, facilitando o paralelismo.

14. Por que sistemas operacionais multitarefa precisam de troca de contexto?

A troca de contexto é o mecanismo que viabiliza a multitarefa em um sistema com um número limitado de CPUs. Para criar a ilusão de execução paralela, o sistema operacional precisa alternar a CPU entre diferentes processos. A troca de contexto é o procedimento de salvar o estado completo de um processo que está saindo da CPU (seu contexto) e carregar o contexto de um novo processo que vai assumir a CPU. Sem esse mecanismo, a CPU ficaria ociosa durante operações de entrada e saída e não seria possível implementar escalonamento preemptivo, limitando a eficiência e a interatividade do sistema.

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

A principal vantagem da memória compartilhada é o desempenho. Por permitir que os processos acessem diretamente uma área comum de memória, ela elimina a sobrecarga de cópias de dados e a mediação do núcleo, tornando-se o método de IPC mais rápido. A grande desvantagem, no entanto, é a complexidade e o risco. Como os processos compartilham o mesmo espaço, a responsabilidade pela sincronização para evitar condições de corrida recai totalmente sobre o programador, que deve implementar mecanismos explícitos como mutexes ou semáforos. Um erro na sincronização pode levar à corrupção de dados.