1)

Em um algoritmo de escalonamento de CPU não preemptivo, uma vez que um processo começa a ser executado, ele não é interrompido até que termine sua execução ou bloqueie voluntariamente. Para determinar o número de maneiras possíveis de organizar a execução de n processos, podemos usar o conceito de permutação.

A permutação de n elementos distintos é dada por n! (n fatorial), que representa o número de maneiras diferentes de organizar esses elementos.
Portanto, para n processos a serem designados para execução em um processador através de um algoritmo não preemptivo, o número de escalonamentos diferentes possíveis é n! (n fatorial).

Se quiser expressar isso em uma fórmula, seria assim:
Número de escalonamentos diferentes = n!
Onde:
n é o número de processos a serem designados para execução.

2)

A diferença fundamental entre o scheduling com preempção e sem preempção está na capacidade de interromper um processo durante sua execução. No primeiro caso, um processo pode ser interrompido por outro de maior prioridade ou que tenha chegado primeiro na fila de escalonamento. Já no segundo caso, um processo só é interrompido ao terminar sua execução ou quando explicitamente solicita uma interrupção.

O scheduling com preempção é prevalente em sistemas operacionais modernos, permitindo que processos de alta prioridade sejam executados rapidamente. Em contrapartida, o scheduling sem preempção pode ser mais eficiente em sistemas com recursos limitados, evitando interrupções de processos de baixa prioridade por processos de alta prioridade.

Os algoritmos de escalonamento variam em relação ao suporte à preempção. Por exemplo, o algoritmo FCFS (First Come, First Served) não suporta preempção, enquanto o SJF (Shortest Job First) pode ser utilizado com ou sem preempção.

Exemplos de sistemas operacionais que usam scheduling com preempção:
Windows, Linux, macOS e Android.

Sistemas utilizam scheduling sem preempção:
Windows 3.1, versões anteriores do macOS antes da versão 8 e alguns sistemas embarcados.

3)

O algoritmo "Shortest Remaining Time First" (SRTF) ou "Shortest Job Next" (SJN) prioriza processos com menor tempo de execução restante. Essa estratégia favorece operações limitadas por E/S, pois processos que frequentemente realizam operações de entrada e saída tendem a liberar a CPU rapidamente, devido aos tempos de CPU mais curtos.

Ao mesmo tempo, esse algoritmo evita a inanição de processos limitados por CPU. Embora dê preferência aos processos com menor tempo de CPU restante, não ignora totalmente os processos

maiores. Mesmo que um processo com maior tempo de execução tenha que esperar brevemente, ele ainda será executado após um curto período de espera.

Dessa forma, ao priorizar os processos com menor tempo de CPU restante, o algoritmo permite que os processos limitados por E/S sejam executados mais prontamente, enquanto ainda oferece oportunidades para os processos mais longos serem executados, evitando a inanição permanente.

4)

É essencial que o escalonador do sistema operacional reconheça a distinção entre programas limitados por I/O e por CPU devido à forma como utilizam os recursos e executam suas tarefas.

Os programas limitados por I/O frequentemente esperam por operações de entrada/saída, enquanto os programas por CPU necessitam intensivamente da CPU para cálculos. Ao reconhecer essa diferença, o escalonador pode aproveitar os momentos de inatividade dos programas I/O-bound para executar outros processos, melhorando a eficiência global da CPU.

Isso não só melhora a resposta do sistema para interações do usuário, mas também evita bloqueios prolongados que poderiam ocorrer se programas por CPU apropriando-se a CPU, prejudicando operações importantes de entrada/saída.

Diferenciar esses tipos de programas permite ao escalonador otimizar a utilização da CPU, melhorar a resposta do sistema e evitar bloqueios prolongados, garantindo eficiência e desempenho superiores do sistema operacional como um todo.

5)

O comando **nice** no Linux permite ajustar a prioridade de um processo. Usuários podem atribuir valores >= 0, mas apenas o usuário root pode atribuir valores < 0. Isso é feito por razões de segurança para evitar abusos, já que valores negativos indicam prioridades mais altas, e permitir que qualquer usuário atribua esses valores poderia resultar em monopolização de recursos ou interferência no desempenho do sistema. Restringir essa capacidade ao **root** ajuda a garantir controle adequado sobre a alocação de recursos do sistema.

6)

O algoritmo de escalonamento mais propenso a resultar em starvation é **"Por prioridades" (d)**, pois processos com prioridades mais baixas podem monopolizar os recursos, impedindo que processos com prioridades mais altas sejam executados.