Nome: Guilherme Bifani de Santana

1 - Um processo é um programa em execução.

2 - Programa é um conjunto de instruções armazenadas (código estático) , enquanto processo é a instância de execução desse programa (entidade dinâmica).

3- Os componentes principais são: o código do programa , o conjunto de dados e o contexto de execução (registradores, stack, etc.).

4- PCB (Process Control Block) é uma estrutura de dados que armazena todas as informações sobre um processo , sendo essencial para o gerenciamento do SO.

5 - É o ato de salvar o estado atual de um processo na CPU (no seu PCB) e carregar o estado (PCB) de outro processo para que este possa ser executado.

6- Os estados possíveis são: Novo , Pronto , Em Execução , Espera/Bloqueado e Terminado.

7- Novo ----------> Pronto: O SO admite o processo.

Pronto ----------> Em execução: O scheduler seleciona o processo.

Em execução ---> Pronto: Expiração do quantum de tempo ou interrupção.

Em execução ---> Espera: Requisição de E/S ou outro evento de bloqueio.

Espera ----------> Pronto: O evento esperado é concluído.

Em execução ---> O processo conclui sua execução.

8- É o conjunto de endereços de memória que um processo pode acessar , incluindo código, dados e stack.

9- É um sistema operacional que mantém múltiplos programas na memória principal simultaneamente , permitindo que a CPU alterne entre eles, aumentando a utilização e o throughput.

10- Gerencia a concorrência usando o escalonamento da CPU (alternando rapidamente a execução) e mecanismos de sincronização (como semáforos e monitores) para acesso seguro a recursos compartilhados.

11- Uma thread é uma unidade básica de execução dentro de um processo. Difere do processo pois threads compartilham o espaço de endereçamento e recursos do processo pai, enquanto processos são independentes e possuem seus próprios recursos.

12- As vantagens incluem maior throughput (produtividade) , melhor responsividade da aplicação , compartilhamento de recursos simplificado e economia de custos na criação/troca de contexto em comparação com processos.

13- Um processo multithread é composto por múltiplos threads , onde cada thread possui seu próprio contador de programa, stack e registradores , mas compartilha o código, dados e arquivos do processo pai.

14- As sub-rotinas são implementadas como threads separadas. O SO ou a biblioteca de threads escalona essas threads para serem executadas, potencialmente em paralelo em CPUs multi-núcleo ou intercaladas em uma única CPU.

15- O principal problema é que o bloqueio de E/S de uma única thread paralisa todo o processo , e não permite a utilização de múltiplos núcleos de CPU para aumentar o desempenho.

16- FORK: Cria um novo processo (filho) que é uma cópia do processo chamador (pai).

JOIN: O processo pai espera a terminação de um ou mais processos filhos criados com FORK antes de continuar sua execução.

17- PARBEGIIN marca o início de um bloco de instruções que podem ser executadas concorrentemente. PAREND marca o fim desse bloco, indicando que o fluxo de controle só deve prosseguir após a conclusão de todas as escuções cocorrentes dentro do bloco.

18- Processos Independentes: Não compartilham recursos com outros processos (exceto por mecanismos de comunicação explícita) e sua execução não afeta outros processos.

19- O principal desafio é que, por não compartilharem espaço de endereçamento, a comunicação exige mecanismos explícitos do SO, como pipes, sockets ou troca de mensagens, que são mais complexos e lentos.

20- Economiza recursos porque a criação e a mudança de contexto de threads são mais rápidas e consomem menos memória e CPU do que a criação e a troca de contexto de processos inteiros , já que threads compartilham a maior parte do contexto do processo.

21- É uma situação onde o resultado da execução de múltiplos processos (ou threads) concorrentes que acessam e modificam dados compartilhados depende da ordem não determinística de suas execuções.

22- É a parte do código de um processo (ou thread) onde ele acessa ou manipula dados ou recursos compartilhados , e que deve ser executada sob exclusão mútua.

23- Semáforos são variáveis inteiras usadas para sinalização. Um semáforo binário (ou mutex) pode ser usado para garantir que apenas um processo entre na região crítica de cada vez (exclusão mútua) , usando operações atômicas wait() e signal().

24- É o mecanismo que permite que um processo (ou thread) se bloqueie e espere até que uma condição específica (relacionada a um dado compartilhado) se torne verdadeira , sendo tipicamente implementada com variáveis de condição e mutexes.

25- Envolve a transferência de dados entre processos usando as operações primitivas enviar(destino, mensagem) e receber(origem, mensagem). O SO é o responsável por intermediar essa troca de dados.

26- Comunicação direta: Os processos devem nomear explicitamente o remetente e o destinatário na operação de envio/recebimento.

Comunicação indireta: A comunicação é feita através de uma entidade intermediária, como uma caixa de correio ou porta. Os processos enviam/recebem da caixa de correio, não um do outro diretamente.

27- Ocorre quando duas ou mais operações tentam modificar o saldo da conta corrente (dado compartilhado) simultaneamente. Se as operações de leitura e escrita não forem atômicas, uma delas pode sobrescrever o resultado da outra, resultando em um saldo incorreto.

28- Se dois processos concorrentes tentam, por exemplo, incrementar a variável compartilhada X (leitura, incremento, escrita), o escalonamento pode fazer com que ambos leiam o mesmo valor antigo de X. Um dos incrementos se perde, pois o segundo processo sobrescreve o resultado do primeiro.

29- Garante que um produtor só tente inserir dados se houver espaço livre no buffer (condição "buffer não cheio") e que um consumidor só tente remover se houver dados (condição "buffer não vazio"). Se a condição não for satisfeita, o processo é bloqueado até que outro processo sinalize a mudança da condição.

30- Alguns mecanismos de software incluem a Solução de Peterson , o uso de variáveis de flag e turn (como no algoritmo de Dekker) , e soluções baseadas em instruções atômicas de hardware (como TestAndSet ou Swap).

31- É uma situação em que dois ou mais processos estão esperando indefinidamente por um recurso que está sendo mantido por outro processo , e nenhum deles pode prosseguir. Ocorre tipicamente em ambientes onde processos competem por recursos.

32- É uma das condições necessárias para deadlock. Ocorre quando um conjunto de processos está em um ciclo de espera, onde cada processo no ciclo está esperando por um recurso que está sendo mantido pelo próximo processo no ciclo.

33- Exclusão Mútua: Os recursos envolvidos são não compartilháveis.

Posse e Espera (Hold and Wait): Um processo pode reter recursos enquanto espera por outros.

Não Preempção: Um recurso não pode ser retirado de um processo à força, deve ser liberado voluntariamente.

Espera Circular: Existe um ciclo de processos, cada um esperando por um recurso mantido pelo próximo.

34- O algoritmo do banqueiro é um algoritmo de prevenção/evitação. Ele garante que o sistema só conceda recursos a um processo se a alocação resultante mantiver o sistema em um "estado seguro" , ou seja, se existir uma sequência de execução que permita que todos os processos terminem.

35- Envolve a manutenção de um grafo de alocação de recursos. O sistema periodicamente verifica o grafo para identificar a existência de ciclos, que indicam a ocorrência de deadlock.

36- As formas de correção (ou recuperação) incluem: Abortar um ou mais processos envolvidos no deadlock e Preempção de recursos de um ou mais processos e alocá-los a outros.

37- Torna-se mais crítico devido ao aumento da concorrência (mais núcleos/CPUs e threads) , e ao maior número de recursos compartilhados (bancos de dados, serviços, etc.) em ambientes distribuídos, aumentando a probabilidade de ocorrência.

38- Para evitar a não preempção (uma das condições de deadlock), o SO pode forçar um processo a liberar um recurso se ele estiver esperando por outro recurso , ou seja, o recurso pode ser temporariamente retirado do processo.

39- Condições de Corrida: Resultados não determinísticos ou incorretos.

Inconsistência de Dados: Dados compartilhados são corrompidos.

Deadlock: Processos ficam bloqueados indefinidamente.

40- A exclusão mútua garante que, a qualquer momento, apenas um processo (ou thread) possa executar sua região crítica. Isso evita que processos concorrentes acessem simultaneamente dados ou recursos compartilhados, prevenindo condições de corrida e inconsistência de dados.