Estratégia Concursos

Aula 00

TJ-RJ (Analista Judiciário - TI - Analista de Infraestrutura) Passo Estratégico de Conhecimentos Específicos

Autor:
Fernando Pedrosa Lopes

26 de Março de 2024

SO: GERENCIAMENTO DE PROCESSOS

Sumário

CONTEÚDO

Sistemas Operacionais e Gerenciamento de Processos. Introdução ao Gerenciamento de Processos. Criação e Eliminação de Processos. Comunicação entre Processos. Sincronização e Concorrência. Escalonamento de Processos. Threads e multithreading.

ANÁLISE ESTATÍSTICA

Inicialmente, convém destacar o percentual de incidência do assunto, dentro da disciplina **Sistemas Operacionais** em concursos/cargos similares. Quanto maior o percentual de cobrança de um dado assunto, maior sua importância.

Obs.: *um mesmo assunto pode ser classificado em mais de um tópico devido à multidisciplinaridade de conteúdo*.

Assunto	Relevância na disciplina em concursos similares
Linux	18.4 %
Windows	11.7 %
1. Windows 10	5.8 %
2. Windows Server 2016	2.2 %
3. Windows Server 2019	1.3 %
4. Windows Server 2012	0.9 %
Virtualização	11.2 %
Cloud Computing	10.3 %
Comandos	9.9 %
Clusters	4.9 %
Sistemas de arquivos	3.1 %
Servidor Web	3.1 %

Sistemas Operacionais	2.7 %
Configuração de rede	1.8 %
Administração de usuários (AD - Active Directory)	1.8 %
Arquitetura de Computadores	1.8 %
Gerência de Memória	1.3 %
Deadlock	1.3 %
Segurança de sistemas operacionais	1.3 %
Conceitos Básicos em Sistemas Operacionais	0.9 %
Gerência do Processador	0.9 %
Apache	0.9 %
Estrutura do Sistema Operacional	0.4 %
Unix	0.4 %
Processos	0.4 %
Kernel	0.4 %
Threads	0.4 %

GLOSSÁRIO DE TERMOS

Faremos uma lista de termos que são relevantes ao entendimento do assunto desta aula. Caso tenha alguma dúvida durante a leitura, esta seção pode lhe ajudar a esclarecer.

Processo: Uma instância de um programa em execução, incluindo o código, dados e estado do programa.

Program Counter: Um registrador que contém o endereço da próxima instrução a ser executada pela CPU.

Registrador: Uma pequena unidade de armazenamento dentro da CPU usada para armazenar dados temporariamente.

Contexto: O estado atual de um processo ou thread, incluindo registradores, program counter, pilha e informações de memória.

CPU Bound: Um processo que gasta a maior parte do seu tempo executando cálculos e faz pouco uso de operações de E/S.

IO Bound: Um processo que gasta a maior parte do seu tempo realizando operações de entrada/saída, como ler ou escrever em arquivos.

Estado Pronto: Estado de um processo que está preparado para executar e aguarda a alocação da CPU.

Estado Em Execução: Estado de um processo que está atualmente sendo executado pela CPU.

Estado Bloqueado: Estado de um processo que está esperando por um evento ou recurso para continuar sua execução.

Fork: Uma chamada de sistema que cria uma cópia exata do processo atual, criando um novo processo filho.

Exec: Uma chamada de sistema que substitui a imagem de memória do processo atual por um novo programa.

Processo Zumbi: Um processo que terminou sua execução, mas ainda tem uma entrada na tabela de processos porque o pai ainda não leu seu status de saída.

Pipe: Um canal de comunicação que permite que processos relacionados troquem dados de maneira sequencial (FIFO).

Socket: Um ponto de extremidade para comunicação entre dois computadores usando uma rede.

Semáforo: Uma variável ou estrutura de dados usada para controlar o acesso a um recurso compartilhado.

Monitor: Uma abstração de sincronização que encapsula variáveis, condições e métodos, fornecendo controle de acesso mutualmente exclusivo.

FCFS (First-Come, First-Served): Um algoritmo de escalonamento que atende processos na ordem em que chegam.

SJN (Shortest Job Next): Um algoritmo de escalonamento que atende o processo com o menor tempo de execução estimado a seguir.

Round Robin: Um algoritmo de escalonamento que atende processos em turnos, dando a cada processo uma quantia fixa de tempo.

Thread: A menor unidade de processamento que pode ser escalonada e executada, geralmente parte de um processo.

Deadlock: Uma situação em que dois ou mais processos estão bloqueados permanentemente, esperando uns pelos outros para liberar recursos.

Livelock: Um estado em que dois ou mais processos continuam ativos mas são incapazes de progredir na execução.

Starvation (Inanição): Uma condição em que um processo é continuamente negado acesso a um recurso e, portanto, incapaz de progredir.

ROTEIRO DE REVISÃO

A ideia desta seção é apresentar um roteiro para que você realize uma revisão completa do assunto e, ao mesmo tempo, destacar aspectos do conteúdo que merecem atenção.

Introdução

Em sistemas operacionais, um processo é uma instância de um programa em execução. Ele consiste em uma sequência de instruções, juntamente com um conjunto de recursos associados, como registradores do processador, contadores de programa, uma pilha, seções de memória e arquivos abertos.

Um processo geralmente é composto pelos seguintes componentes:

- **Texto do Programa**: A sequência real de instruções.

- **Contador de Programa**: Um registrador que aponta para a próxima instrução a ser executada.
- **Registradores**: Valores atuais dos registradores do CPU.
- **Pilha**: Contém dados temporários, como endereços de retorno de chamadas de função.
- **Estado do Processo**: Pode estar em um dos vários estados, como novo, pronto, em execução, esperando, terminado, etc.

O **contexto** de um processo refere-se ao estado de um processo em um determinado momento, incluindo todos os valores dos registradores do processador, a área de memória que ele ocupa, etc. O contexto é salvo e restaurado para permitir que o sistema operacional troque processos dentro e fora do CPU, uma prática conhecida como comutação de contexto.

Em sistemas operacionais existem basicamente dois níveis de processos:

- **Processos de Usuário**: Programas executados por usuários.
- **Processos de Sistema**: Executados pelo próprio sistema operacional.

Classificação de Processos

Os processos podem ser classificados de acordo com o tipo de processamento que realizam. Sob esse aspecto, pode haver dois tipos de processos:

CPU-bound

Um processo é dito CPU-bound (ligado à CPU) quando passa a maior parte do tempo no estado de execução, ou seja, utilizando o processador. Esse tipo de processo realiza poucas operações de entrada/saída e é encontrado em aplicações matemáticas ou científicas, que efetuam muitos cálculos e poucas operações de leitura/gravação.

I/O-bound

Um processo é chamado I/O-bound (ligado à E/S) quando passa a maior parte do tempo no estado de espera, pois realiza um elevado número de operações de entrada/saída. Esse tipo de processo é encontrado em aplicações comerciais, que se baseiam em leitura, processamento e gravação. Os processos interativos também são bons exemplos de processos I/O-bound, pela forma de comunicação entre o usuário e o sistema, normalmente lenta, devido à utilização de terminais.

Estados de Processos

Estados de processos em um sistema operacional descrevem as várias fases ou condições que um processo pode experimentar durante seu ciclo de vida. Os estados de processo são fundamentais para o gerenciamento e escalonamento adequados dos processos. Veja os principais:

1. Estado Novo

- **Descrição**: O processo foi criado, mas ainda não foi admitido na pool de processos prontos.
- **Transição**: Quando o sistema operacional aceita o novo processo, ele muda para o estado "Pronto".

2. Estado Pronto

- **Descrição**: O processo está pronto para ser executado e aguarda a atribuição do processador.
- **Transição**: O escalonador pode selecionar o processo para executar, movendo-o para o estado "Em Execução".

3. Estado Em Execução

- **Descrição**: O processo está atualmente sendo executado no processador.
- **Transições**: O processo pode ser movido de volta para o estado "Pronto" se outro processo receber o CPU, ou para o estado "Esperando" se ele precisa esperar por um evento.

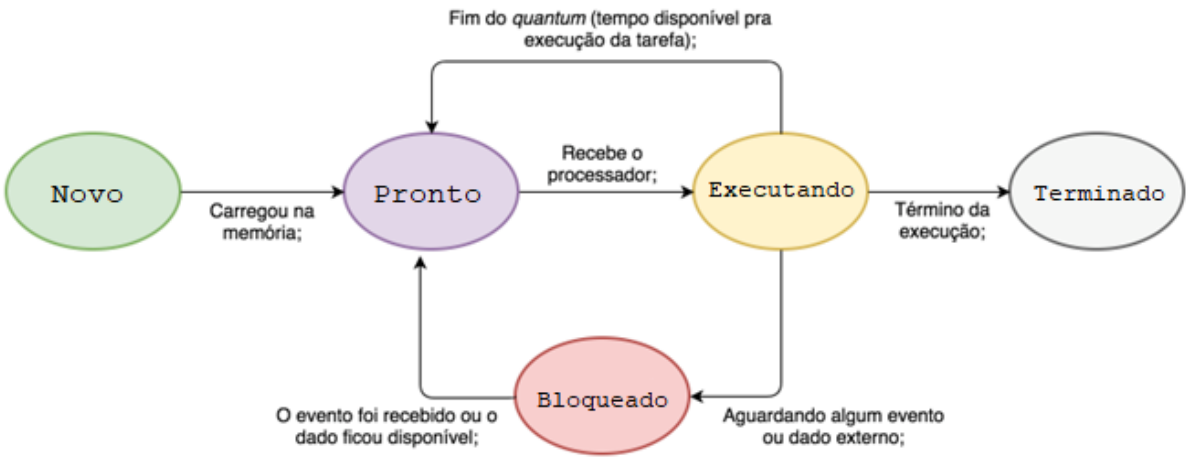
4. Estado Bloqueado ou Suspenso

- **Descrição**: O processo está esperando por algum evento, como a conclusão de uma operação de E/S.
- **Transição**: Uma vez que o evento ocorra, o processo volta ao estado "Pronto".

5. Estado Terminado

- **Descrição**: O processo concluiu sua execução.
- **Transição**: Este é o estado final, e o processo é removido da memória.

Os estados acima podem ser visualizados em um diagrama de transição de estados, mostrando como os processos se movem entre os diferentes estados:



A **comutação** entre esses estados envolve salvar e restaurar o contexto do processo, que inclui todos os detalhes necessários para retomar o processo a partir de onde parou.

Concorrência

O conceito de concorrência é fundamental para compreendermos o funcionamento de um sistema operacional multiprogramável. A possibilidade de periféricos e dispositivos funcionarem simultaneamente entre si, juntamente com a CPU, permitiu a execução de tarefas concorrentes, que é o princípio básico para projeto e implementação de sistemas multiprogramáveis. Sistemas operacionais podem ser vistos como um conjunto de rotinas que executam concorrentemente de uma forma ordenada.

Os sistemas multiprogramáveis surgiram de um problema existente nos sistemas monoprogramáveis, que é a baixa utilização de recursos do sistema, como processador, memória e periféricos.

Nos sistemas monoprogramáveis, somente um programa pode estar residente na memória, e a CPU permanece dedicada, exclusivamente, à execução desse programa. Podemos observar que, nesse tipo de sistema, ocorre um desperdício na utilização da CPU (CPU - Unidade Central de Processamento), pois enquanto o programa está realizando, por exemplo, uma leitura em disco, o processador permanece sem realizar nenhuma tarefa. O tempo de espera é consideravelmente grande, já que as operações com dispositivos de entrada e saída são muito lentas se comparadas com a velocidade da CPU.

Outro aspecto que devemos considerar é a subutilização da memória. Um programa que não ocupe totalmente a memória principal, ocasiona a existência de áreas livres, sem utilização.

Nos sistemas multiprogramáveis, vários programas podem estar residentes em memória, concorrendo pela utilização da CPU. Dessa forma, quando um programa solicita uma operação de entrada/saída, outros programas poderão estar disponíveis para utilizar o processador. Nesse caso, a CPU permanece menos tempo ociosa e a memória principal é utilizada de forma mais eficiente, pois existem vários programas residentes se revezando na utilização do processador.

A utilização concorrente da CPU deve ser implementada de maneira que, quando um programa perde o uso do processador e depois retoma para continuar o processamento, seu estado deve ser idêntico ao do momento em que foi interrompido. O programa deverá continuar sua execução exatamente na instrução seguinte àquela em que havia parado, aparentando ao usuário que nada aconteceu. Em sistemas de tempo compartilhado, existe a impressão de que o computador está inteiramente dedicado ao usuário, ficando todo esse mecanismo transparente para ele. No caso de periféricos, é comum termos, em sistemas monoprogramáveis, impressoras paradas por um grande período de tempo e discos com acesso restrito a um único usuário. Esses problemas são solucionados em sistemas multiprogramáveis, onde é possível compartilhar impressoras entre vários usuários e realizar acesso concorrente a discos por diversos programas.

Criação e Eliminação de Processos

Criação de Processos Filhos

Um processo pode criar um ou mais processos durante sua execução. Esses processos recém-criados são chamados de processos filhos. Eles são criados por um processo pai e podem, por sua vez, criar seus próprios processos filhos, formando uma árvore de processos.

Veja como os Processos Filhos são Criados:

- **Fork**: O sistema operacional fornece uma chamada de sistema, geralmente denominada "fork", que permite a um processo criar uma cópia exata de si mesmo. O processo original é chamado de processo pai, e a cópia é o processo filho.
- **Exec**: Após o fork, o processo filho pode substituir sua imagem de processo pela de um programa diferente usando uma chamada de sistema como "exec". Isso permite que o processo filho execute um programa diferente do processo pai.

Quando um processo filho é criado, geralmente ele carrega os seguintes atributos:

- **ID do Processo (PID)**: Cada processo filho recebe um PID único.
- **Contexto**: O processo filho herda o contexto do processo pai, incluindo registradores, variáveis, permissões, etc.
- **Estado**: Inicialmente, o processo filho estará no estado "Pronto" e aguardará sua vez de ser executado.

Os processos pai e filho podem se comunicar através de vários mecanismos, como pipes, memória compartilhada ou arquivos. Dois usos comuns para a criação de processos filhos são a **execução paralela**, que permite que tarefas sejam executadas simultaneamente e **isolamento de tarefas**, que permite separar tarefas em processos diferentes para segurança e robustez.

Veja como você pode criar processos filhos em C:

```c
#include <unistd.h>

int main() {
  pid_t pid = fork(); // Cria um processo filho

  if (pid == 0) {
    // Código do processo filho
    execvp("program", args); // Substitui a imagem do processo
  } else {
    // Código do processo pai
  }

  return 0;
}
```

Finalização de Processos

A finalização de processos refere-se ao procedimento pelo qual um processo conclui sua execução e é removido do sistema. A seguir, veja os principais aspectos da finalização de processos.

Finalização Normal

Um processo pode ser terminado de maneira normal, geralmente devido à conclusão bem-sucedida de sua tarefa. Isso pode ocorrer de várias maneiras:

- **Retorno do Programa**: O processo retorna de sua função principal, liberando os recursos e notificando o sistema operacional.
- **Chamada de Saída**: O processo invoca uma chamada de sistema explícita, como **exit()**, para terminar sua execução.
- **Terminação pelo Pai**: Um processo pai pode terminar um de seus processos filhos usando uma chamada de sistema como **wait()**.

Finalização Anormal ou Forçada

Às vezes, um processo pode ser forçado a terminar devido a circunstâncias excepcionais:

- **Erro Fatal**: Se um processo encontra um erro que não pode ser recuperado.
- **Interrupção pelo Usuário**: Um usuário ou administrador pode forçar a terminação de um processo.
- **Terminação por Outro Processo**: Um processo pode ser terminado por outro processo através de uma chamada de sistema, como **kill()**.

Consequências da Terminação

Quando um processo é finalizado, várias ações são realizadas pelo sistema operacional:

- **Liberação de Recursos**: Todos os recursos alocados ao processo, como memória e arquivos abertos, são liberados.
- **Remoção da Tabela de Processos**: O processo é removido das estruturas de dados do sistema operacional, como a tabela de processos.
- **Comunicação com Processos Relacionados**: Se o processo tiver processos filhos ou outros processos dependentes, medidas apropriadas devem ser tomadas, como terminar processos filhos ou notificar outros processos.

Processos Zumbis

Se um processo filho termina, mas o processo pai não reconhece a terminação (por exemplo, através de uma chamada **wait()**), o processo filho pode permanecer no sistema como um "processo zumbi". Isso pode levar a desperdício de recursos.

A finalização de processos deve ser gerenciada com cuidado para evitar vazamentos de recursos, inconsistências e outros problemas potenciais. A coordenação e comunicação adequadas entre processos relacionados são vitais durante a terminação.

Comunicação e Sincronização entre Processos

A comunicação entre processos (IPC, do inglês Inter-Process Communication) é fundamental para permitir que processos cooperativos em um sistema operacional compartilhem informações, coordenem atividades e alcancem objetivos comuns. Vejamos os aspectos principais da comunicação entre processos.

Canais de Comunicação

Os canais de comunicação são os meios pelos quais os processos trocam dados. Existem vários métodos:

- **Pipes (Canais)**:
    1. **Pipes Nomeados**: Também conhecidos como FIFOs, permitem a comunicação entre processos não relacionados.
    2. **Pipes Anônimos**: Utilizados para comunicação entre processos pai e filho.
- **Memória Compartilhada**:
    1. Permite que vários processos acessem uma região de memória comum, facilitando a troca rápida de informações.
    2. Pode exigir mecanismos de sincronização para evitar conflitos.
- **Filas de Mensagens**:
    1. Permitem que os processos enviem e recebam mensagens de uma fila comum.
    2. Oferecem flexibilidade e podem ser usadas para comunicação entre processos não relacionados.
- **Sockets**:
    1. Utilizados para comunicação entre processos em diferentes máquinas em uma rede.
    2. Podem ser usados localmente para comunicação entre processos na mesma máquina.

Sincronização de Processos

Existem três fatores fundamentais na sincronização de processos concorrentes que deverão ser atendidos:

1. O número de processadores e o tempo de execução dos processos concorrentes devem ser irrelevantes;

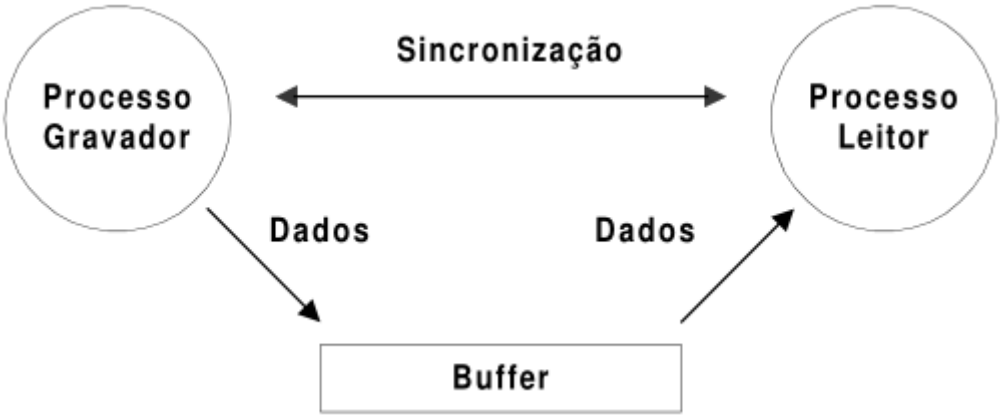
2. Um processo, fora de sua região crítica, não pode impedir que outros processos entrem em suas próprias regiões críticas;
3. Um processo não pode permanecer indefinidamente esperando para entrar em sua região crítica.

A sincronização de processos é necessária para coordenar o acesso e a execução de processos que compartilham recursos ou dependem uns dos outros. Os principais mecanismos incluem:

- **Semáforos**:
    1. Variáveis especiais usadas para controlar o acesso a recursos compartilhados.
    2. Pode ser usado para resolver problemas clássicos de sincronização, como o problema do produtor-consumidor.
- **Monitores**:
    1. Estruturas de alto nível que encapsulam dados compartilhados e procedimentos.
    2. Oferecem uma maneira mais abstrata e segura de sincronizar processos.
- **Variáveis de Condição**:
    1. Usadas em conjunto com monitores para permitir que os processos esperem por uma condição específica.
    2. Facilitam a coordenação precisa entre processos.
- **Barreiras**:
    1. Utilizadas para fazer com que um conjunto de processos espere até que todos alcancem um certo ponto, facilitando a execução em fases.

A comunicação e sincronização eficientes entre processos são vitais para o desempenho e a confiabilidade dos sistemas multitarefa e multiprocessador. A escolha do canal de comunicação e o mecanismo de sincronização apropriados dependem das necessidades específicas da aplicação, visto que a sincronização incorreta pode levar a problemas complexos, como deadlocks e condições de corrida.

Vejamos um exemplo onde dois processos concorrentes trocam informações através de operações de gravação e leitura em um buffer. Um processo só poderá gravar dados no buffer caso ele não esteja cheio. Da mesma forma, um processo só poderá ler dados armazenados do buffer se existir algum dado para ser lido. Em ambos os casos, os processos deverão aguardar até que o buffer esteja pronto para as operações de gravação ou de leitura



Semáforos

Vamos nos aprofundar um pouco mais sobre o tópico de semáforos, por ser um aspecto importante da sincronização entre processos.

Um semáforo é uma variável inteira que pode ser usada como sinalizador ou contador, representando a disponibilidade de recursos ou o estado de um processo. Ele oferece duas operações atômicas fundamentais: **wait()** (P) e **signal()** (V):

- **wait(semáforo)**:
  - Se o valor do semáforo for maior que zero, decrementa o valor.
  - Se o valor for zero, o processo chamador é bloqueado e colocado em uma fila de espera.
- **signal(semáforo)**:
  - Incrementa o valor do semáforo.
  - Se houver processos bloqueados, um deles é despertado e movido para o estado "Pronto".

Existem basicamente dois tipos de semáforos:

- **Semáforos Binários**:
  1. Valor limitado a 0 ou 1.
  2. Usados como travas (locks) para controlar o acesso exclusivo a recursos.
- **Semáforos de Contagem**:
  1. Valor pode ser qualquer número inteiro positivo.
  2. Usados para representar a disponibilidade de múltiplos recursos idênticos ou para coordenar a ordem de execução.

Semáforos são aplicados, principalmente, na sincronização de processos e uso de recursos compartilhados. Veja:

Controle de Acesso a Recursos Compartilhados:

- Semáforos podem ser usados para garantir que apenas um processo por vez tenha acesso a um recurso compartilhado.

Solução de Problemas Clássicos de Sincronização:

- **Problema do Produtor-Consumidor**: Semáforos coordenam produtores que produzem itens e consumidores que consomem itens de um buffer compartilhado.
- **Problema do Jantar dos Filósofos**: Semáforos são usados para evitar impasses entre filósofos competindo por garfos compartilhados.

A utilização incorreta de semáforos pode levar a deadlocks, onde processos esperam uns pelos outros indefinidamente. Por fim, semáforos ajudam a evitar condições de corrida, onde a saída depende da ordem de execução dos processos.

Veja como você pode simular a implementação de semáforos em C:

```c
#include <semaphore.h>

sem_t semaphore;

// Processo que utiliza o recurso
void process() {
  sem_wait(&semaphore); // Espera pelo recurso
  // Acesso ao recurso compartilhado
  sem_post(&semaphore); // Libera o recurso
}
```

Problemas Clássicos de Sincronização

Problemas clássicos de sincronização são cenários teóricos que demonstram desafios comuns em coordenação e sincronização de processos em sistemas operacionais. Esses problemas são estudados para entender os princípios da concorrência e desenvolver técnicas para lidar com a competição por recursos compartilhados. Vamos detalhar alguns dos problemas clássicos de sincronização:

1. Problema do Produtor-Consumidor

- **Descrição**: Existem dois tipos de processos, produtores e consumidores, que compartilham um buffer comum. Os produtores adicionam itens ao buffer, e os consumidores removem itens.
- **Desafio**: Sincronizar produtores e consumidores para que os consumidores não consumam quando o buffer estiver vazio e os produtores não produzam quando o buffer estiver cheio.
- **Solução Comum**: Utilizar semáforos ou monitores para controlar o acesso ao buffer.

2. Problema do Jantar dos Filósofos

- **Descrição**: Cinco filósofos sentados em uma mesa, cada um compartilhando um garfo com o vizinho. Para comer, um filósofo precisa de ambos os garfos adjacentes.
- **Desafio**: Evitar que os filósofos entrem em impasse (deadlock), onde cada um pega um garfo e espera pelo outro, ou em inanição (starvation), onde um filósofo é continuamente impedido de comer.
- **Solução Comum**: Utilizar semáforos ou impor uma ordem na aquisição dos garfos.

3. Problema dos Leitores-Escritores

- **Descrição**: Vários processos precisam acessar uma base de dados compartilhada. Alguns são leitores e outros são escritores. Leitores só leem a base de dados, enquanto escritores podem ler e modificar.
- **Desafio**: Permitir múltiplos leitores simultâneos, mas garantir acesso exclusivo para escritores.
- **Solução Comum**: Utilizar semáforos, monitores ou locks com prioridades variáveis.

4. Problema do Barbeiro Dorminhoco

- **Descrição**: Um barbeiro em uma barbearia com um número limitado de cadeiras. Se não houver clientes, o barbeiro dorme. Se um cliente chega e todas as cadeiras estão ocupadas, ele vai embora.
- **Desafio**: Sincronizar barbeiro e clientes para que o barbeiro durma quando não houver clientes e seja acordado quando um cliente chegar.
- **Solução Comum**: Utilizar semáforos para representar o estado da barbearia e coordenar a ação entre barbeiro e clientes.

Escalonamento de Processos

Escalonamento de processos é o método pelo qual os processos são selecionados para execução, à ordem em que são executados e ao tempo que lhes é alocado. Os seu principais objetivos são:

- **Justiça**: Garantir que cada processo receba uma parcela justa do tempo de CPU.
- **Eficiência**: Maximizar a utilização da CPU e minimizar o tempo de resposta.
- **Throughput**: Maximizar o número de processos concluídos por unidade de tempo.
- **Resposta Rápida**: Minimizar o tempo de resposta para processos interativos.
- **Priorização**: Permitir que processos mais importantes sejam atendidos antes.

Existem diferentes tipos de escalonamento, dentre os quais podemos citar:

- **Escalonamento de Longo Prazo**: Decide quais processos são admitidos na fila de processos prontos. Controla o grau de multiprogramação.
- **Escalonamento de Médio Prazo**: Pode suspender ou retomar processos para equilibrar recursos.
- **Escalonamento de Curto Prazo (CPU Scheduling)**: Decide qual processo na fila de processos prontos será executado a seguir. É o mais frequentemente discutido e se refere aos algoritmos de escalonamento que veremos a seguir.

Lembrando que processos podem ser preemptivos ou não preemptivos. No **Escalonamento Preemptivo**, a CPU pode ser retirada de um processo antes de sua conclusão, permitindo uma resposta mais rápida às solicitações. Já no **Escalonamento Não Preemptivo**, a CPU é alocada a um processo até sua conclusão ou liberação voluntária.

Finalmente, é importante destacarmos o conceito de **reentrância**.

É comum, em sistemas multiprogramáveis, vários usuários executarem os mesmos utilitários do sistema operacional simultaneamente, como, por exemplo, um editor de textos. Se cada usuário que utilizasse o editor trouxesse o código do utilitário para a memória, haveria diversas cópias de um mesmo programa na memória principal, o que ocasionaria um desperdício de espaço.

Reentrância é a capacidade de um código de programa (código reentrante) poder ser compartilhado por diversos usuários, exigindo que apenas uma cópia do programa esteja na memória. Uma característica da reentrância é que o código não pode ser modificado por nenhum usuário no momento em que está sendo executado.

A reentrância permite que cada usuário possa estar em um ponto diferente do código reentrante, manipulando dados próprios, exclusivos de cada usuário. Os utilitários do sistema, como editores

de texto, compiladores e linkers, são exemplos de código reentrante, que proporciona grande economia de espaço em memória e aumento na performance do sistema.

Algoritmos de Escalonamento

As políticas de escalonamento, ou algoritmos de escalonamento, são conjuntos de regras que o sistema operacional utiliza para determinar a ordem na qual os processos na fila de prontos são selecionados para execução. Essas políticas têm impacto direto em vários aspectos do desempenho do sistema, como eficiência, tempo de resposta e justiça. Veja algumas das políticas de escalonamento mais comuns:

1. First-Come, First-Served (FCFS)

- **Descrição**: Processos são atendidos na ordem em que chegam à fila de prontos.
- **Vantagens**: Simples, fácil de implementar.
- **Desvantagens**: Pode levar a *starvation* e ao efeito de "comboio" se processos longos precederem processos curtos. Esse efeito é nomeado em analogia a um comboio (convoy) de navios ou veículos que são forçados a viajar à velocidade do membro mais lento.

2. Shortest Job Next (SJN)

- **Descrição**: O próximo processo selecionado é aquele com o menor tempo de execução estimado.
- **Vantagens**: Ótimo para minimizar o tempo de espera.
- **Desvantagens**: Pode ser difícil estimar o tempo de execução; processos longos podem ser inanidos (morrer de fome, *starvation*).

3. Round Robin (RR)

- **Descrição**: Cada processo recebe uma quantia fixa de tempo (quantum) em turnos.
- **Vantagens**: Justo e responsivo.
- **Desvantagens**: A escolha do quantum afeta o desempenho; pode não ser ideal para processos de diferentes tamanhos.

4. Escalonamento com Prioridades

- **Descrição**: Processos são atendidos com base em um nível de prioridade.
- **Vantagens**: Flexível, pode atender a requisitos específicos.
- **Desvantagens**: Pode levar à inanição de processos de baixa prioridade; problemas de envelhecimento.

5. Escalonamento de Múltiplos Níveis com Feedback

- **Descrição**: Utiliza várias filas com diferentes prioridades e regras de escalonamento.
- **Vantagens**: Adaptável a diferentes tipos de processos.
- **Desvantagens**: Mais complexo de implementar e gerenciar.

6. Escalonamento em Tempo Real

- **Descrição**: Utilizado em sistemas de tempo real onde as tarefas têm prazos rígidos.
- **Vantagens**: Adequado para sistemas críticos.
- **Desvantagens**: Exige análise cuidadosa e design preciso.

Ao escolher uma política de escalonamento, é importante levar em conta um conjunto de características, tais como:

- **Natureza dos Processos**: Processos interativos, em lote, de tempo real, etc., podem exigir diferentes políticas.
- **Objetivos do Sistema**: Maximizar o throughput, minimizar o tempo de resposta, garantir a justiça, etc.
- **Recursos Disponíveis**: A quantidade e o tipo de recursos podem influenciar a política escolhida.
- **Preempção**: Algumas políticas permitem que a CPU seja retirada de um processo (preemptivo), enquanto outras não (não preemptivo).

Threads e Multithreading

Threads, também conhecidas como linhas de execução, são uma forma de permitir a execução concorrente dentro de um único processo. Elas representam a menor unidade de processamento que pode ser agendada e executada por um sistema operacional.

Uma thread é uma sequência de instruções executadas dentro de um processo. Múltiplas threads dentro de um processo compartilham o mesmo espaço de endereçamento e recursos, como arquivos abertos e variáveis globais. Cada thread possui seu próprio contexto, incluindo registradores, contador de programa e pilha.

Vantagens de Utilizar Threads:

- **Eficiência**: A criação e troca de contexto entre threads é geralmente mais rápida do que entre processos, pois elas compartilham muitos recursos.

- **Compartilhamento de Recursos**: Threads de um mesmo processo podem compartilhar dados e recursos facilmente, facilitando a comunicação e colaboração.
- **Responsividade**: Em aplicações interativas, o uso de threads pode melhorar a responsividade, permitindo que uma parte do programa continue executando enquanto outra parte está esperando por E/S.

Tipos de Threads:

1. **Threads de Usuário (User-Level Threads)**:
    - Gerenciadas inteiramente pela biblioteca de threads no espaço do usuário.
    - O kernel não tem conhecimento dessas threads, o que as torna rápidas de criar e gerenciar.
    - A desvantagem é que uma chamada de sistema bloqueante pode bloquear todas as threads do processo.
2. **Threads de Kernel (Kernel-Level Threads)**:
    - Suportadas e gerenciadas diretamente pelo sistema operacional.
    - Oferecem mais funcionalidades e são mais robustas, mas têm um custo de gerenciamento mais alto.
    - Permitem o verdadeiro paralelismo em sistemas multiprocessadores.

Modelos de Multithreading

Modelos de multithreading descrevem como as threads no espaço do usuário são mapeadas para as threads no espaço do kernel (sistema operacional). Esses modelos têm impacto direto na maneira como as threads são gerenciadas e na eficiência e flexibilidade do sistema. A seguir, vamos ver os três principais modelos de multithreading:

1. Muitas para Um (Many-to-One)

- **Descrição**: Neste modelo, várias threads de usuário são mapeadas para uma única thread de kernel.
- **Vantagens**:
    - **Gerenciamento Eficiente**: As threads de usuário são leves e rápidas de criar e destruir.
    - **Portabilidade**: Como o gerenciamento é feito no espaço do usuário, esse modelo pode ser usado em diferentes sistemas operacionais.
- **Desvantagens**:
    - **Bloqueio Global**: Se uma thread de usuário fizer uma chamada de sistema bloqueante, todas as outras threads do mesmo processo serão bloqueadas, já que há apenas uma thread de kernel.

- **Sem Paralelismo Real**: Como há apenas uma thread de kernel, o processador não pode executar várias threads do mesmo processo simultaneamente.

2. Um para Um (One-to-One)

- **Descrição**: Cada thread de usuário é mapeada para uma thread de kernel dedicada.
- **Vantagens**:
  - **Paralelismo Real**: Várias threads de um processo podem ser executadas simultaneamente em múltiplos processadores.
  - **Isolamento**: Uma chamada de sistema bloqueante em uma thread não afeta as outras threads do processo.
- **Desvantagens**:
  - **Custo de Criação e Gerenciamento**: A criação e o gerenciamento de threads de kernel são mais pesados, o que pode limitar o número de threads.
  - **Possível Sobrecarga**: Se houver muitas threads, o sistema operacional pode gastar muito tempo apenas gerenciando-as, afetando o desempenho geral.

3. Muitas para Muitas (Many-to-Many)

- **Descrição**: Múltiplas threads de usuário são mapeadas para um conjunto (que pode ser menor ou igual) de threads de kernel.
- **Vantagens**:
  - **Flexibilidade**: Permite que o sistema equilibre as vantagens dos modelos Many-to-One e One-to-One.
  - **Paralelismo com Eficiência**: Pode executar threads em paralelo, como o One-to-One, mas com menos sobrecarga.
  - **Isolamento de Chamadas de Sistema Bloqueantes**: Uma chamada de sistema bloqueante em uma thread de usuário não bloqueia necessariamente todas as outras threads do processo.
- **Desvantagens**:
  - **Complexidade**: É um modelo mais complexo para implementar e gerenciar, exigindo coordenação cuidadosa entre o espaço do usuário e do kernel.

Deadlock, Livelock e Starvation

Deadlock, livelock e starvation são três condições indesejadas que podem ocorrer em sistemas concorrentes, onde múltiplos processos ou threads competem por recursos. Esses fenômenos

podem levar a sistemas instáveis, ineficientes ou não responsivos. Vejamos cada um desses conceitos:

Deadlock (Impasse)

Deadlock ocorre quando dois ou mais processos estão bloqueados permanentemente, cada um esperando que o outro libere um recurso, ou mais genericamente, cumpra uma condição, para que possa continuar.

Para que um deadlock ocorra, quatro condições devem ser satisfeitas simultaneamente:

1. **Exclusão Mútua**: Cada recurso é ou não compartilhável.
2. **Espera e Posse (Hold and Wait)**: Um processo segurando pelo menos um recurso está esperando por recursos adicionais.
3. **Não Preempção**: Recursos não podem ser forçadamente retirados de um processo.
4. **Espera Circular**: Existe um ciclo de processos onde cada processo está esperando por um recurso que o próximo processo no ciclo detém.

Algumas abordagens para lidar com deadlocks incluem:

- **Prevenção**: Garantir que pelo menos uma das quatro condições necessárias nunca ocorra.
- **Detecção e Recuperação**: Permitir que o deadlock ocorra, detectá-lo e então tomar medidas para recuperar o sistema.

Livelock

Livelock é uma situação em que dois ou mais processos estão bloqueados em uma lógica de execução, onde cada processo está tentando evitar o conflito, mas na verdade está impedindo o progresso mútuo.

Exemplo:

Dois processos estão tentando acessar um recurso e, ao detectar o conflito, ambos recuam e tentam novamente, entrando em um ciclo infinito de tentativas sem progresso.

Solução:

- Introduzir aleatoriedade ou prioridades nas tentativas de acesso para que um dos processos tenha preferência.

Starvation (Inanição)

Starvation ocorre quando um ou mais processos são impedidos de acessar um recurso por um período indefinido, enquanto outros processos continuam a ser servidos.

Causas:

- **Políticas de Escalonamento Injustas**: Por exemplo, um algoritmo de escalonamento de prioridades que sempre favorece processos de alta prioridade pode levar à inanição de processos de baixa prioridade.
- **Concorrência por Recursos**: Processos que precisam de muitos recursos podem ser continuamente preteridos por processos que precisam de menos recursos.

Solução:

- Implementar mecanismos de envelhecimento, onde a prioridade de um processo aumenta com o tempo de espera.
- Utilizar algoritmos de escalonamento que garantem uma alocação justa de recursos.

QUESTÕES ESTRATÉGICAS

Nesta seção, apresentamos e comentamos uma amostra de questões objetivas selecionadas estrategicamente: são questões com nível de dificuldade semelhante ao que você deve esperar para a sua prova e que, em conjunto, abordam os principais pontos do assunto.

A ideia, aqui, não é que você fixe o conteúdo por meio de uma bateria extensa de questões, mas que você faça uma boa revisão global do assunto a partir de, relativamente, poucas questões.

1. **(FCC/Câmara Municipal-SP - 2014)** No escalonamento usando o algoritmo Round-Robin,

A) o escalonador seleciona o processo à espera com o menor tempo de execução estimado até a conclusão, reduzindo o tempo médio de espera, mas aumentando a variância dos tempos de resposta.

B) processos são despachados na ordem FIFO (First-in-First-Out), mas recebem uma quantidade limitada de tempo de processador denominada quantum.

C) a prioridade de cada processo é uma função não apenas do seu tempo de serviço, mas também do tempo que passou esperando pelo serviço.

D) o escalonador ajusta dinamicamente o comportamento do processo, de tal forma que o próximo processo a obter o processador seja aquele que chegar à frente da fila de nível mais alto, que não estiver vazia, na rede de filas.

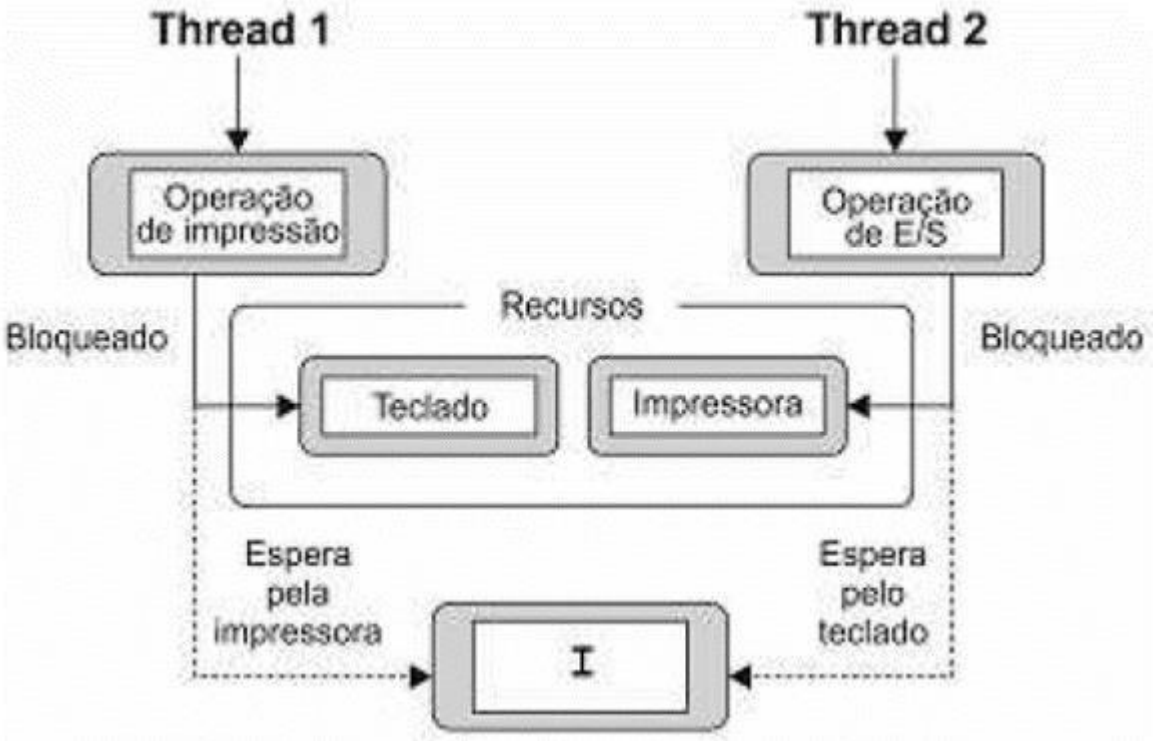
E) o processo que tem o prazo de execução mais curto é favorecido, medindo a diferença entre o tempo que um processo requer para finalizar e o tempo restante até atingir o seu prazo final.

Comentários:

Algoritmo Round-robin: é realizado um rodízio entre os processos, sendo que a cada processo é atribuído um intervalo de tempo (quantum), durante o qual ele pode ser executado. Se ao final do quantum o processo ainda estiver em execução é realizada a preempção da CPU e esta é alocada a um outro processo. Obviamente que se o processo tiver terminado antes do quantum ter expirado ou se tiver sido bloqueado, a troca da CPU é realizada neste momento. Portanto, a alternativa B está correta e é o gabarito da questão.

Gabarito: Letra B

2. **(FCC/DPE-RS - 2017)** Considere a figura abaixo.



Do ponto de vista do sistema operacional, a situação indica que a caixa I deve ser preenchida com?

A) starvation.

B) multithreading.

C) superthreading.

D) deadlock.

E) hyperthreading.

Comentários:

Note que a thread 1 está realizando uma operação de impressão, bloqueando o teclado e está à

espera da impressora. A thread 2 está realizando uma operação de E/S, bloqueando a impressora e à espera do teclado. Ou seja, nenhuma thread libera o recurso que está usando e cada uma quer um recurso que a outra possui. As duas ficarão “trancadas”, esperando... isso é um deadlock! Portanto, a alternativa D está correta e é o gabarito da questão.

Gabarito: Letra D

3. **(FCC/DPE-RS - 2017)** Dentre as políticas de escalonamento de processos a seguir, a que apresenta maior probabilidade de ocasionar o starvation é a

A) Round Robin.
B) de tempo compartilhado.
C) First In First Out.
D) preemptiva.
E) não preemptiva.

Comentários:
Uma política de escalonamento não-preemptiva é aquela que um recurso não pode ser retirado de um processo, a não ser que "ele queira". Imagine um processo que comece a utilizar a CPU e fique utilizando por 1h. Além disso digamos que há uma certa prioridade entre os processos e existe uma demanda muito grande pelo uso da CPU. Pode ocorrer que um processo com prioridade baixa jamais seja executado! Ou seja, starvation! Portanto, a alternativa E está correta e é o gabarito da questão.

Gabarito: Letra E

4. **(IF-PA/IF-PA - 2019)** Em relação à gerência de processo, marque a alternativa CORRETA:

A) o processo é um programa em no estado de pronto.
B) os estados do processo são (execução, pronto, bloqueado ou espera).
C) a thread permite que apenas uma execução ocorra no mesmo ambiente do processo.
D) os sinais são mecanismos que permitem notificar o sistema operacional de eventos gerados pelo processador.
E) escalonamento é a escolha do processo, em estado de execução.

Comentários:

(A) Processo é um programa em execução; (B) Melhor ver na figura abaixo; (C) Thread permite que fluxos de execução em um mesmo processo ocorram quase que em paralelo; (D) Interrupções são mecanismos que permitem notificar o sistema operacional de eventos gerados pelo processador; (E) escalonamento é a escolha do processo, em estado de pronto.

Obs.: Bloqueado ou em espera é a mesma coisa! Portanto, a alternativa B está correta e é o gabarito da questão.

Gabarito: Letra B

5. **(IESES/IGP-SC - 2017)** Acerca da gerência de processos dos sistemas operacionais, assinale a alternativa correta:

A) Um conjunto de processos está em estado de deadlock quando todos os processos no conjunto estão esperando por um evento que só pode ser causado por outro processo do conjunto.
B) Em um escalonamento preemptivo, um processo só perde o processador se terminar ou entrar em estado de espera.
C) No algoritmo de escalonamento de processos Round Robin, o escalonador sempre escolhe para execução o processo com menor expectativa de tempo de processamento. Esse algoritmo baseia-se no fato de que privilegiando processos pequenos o tempo médio de espera decresce.
D) Starvation é uma situação que não pode ocorrer quando um sistema operacional provê prioridades a processos.

Comentários:

(A) Isso aí... fica um aguardando o outro liberar um recurso, que por sua vez aguarda um terceiro e por aí vai... (B) Em um escalonamento preemptivo um processo perde o processador se terminar, ficar bloqueado ou se terminar seu quantum; (C) No algoritmo de escalonamento de processos Round Robin ocorre um "rodízio", sendo que cada processo recebe a mesma quantidade de quantum em uma ordem FIFO; (D) Starvation justamente ocorre quando um sistema operacional provê prioridades a processos. Imagine o uso do recurso processador, sendo que um processo tenha a prioridade máxima e um outro tenha a mínima. O primeiro leve 4 dias para terminar a execução e antes disso outros processos "surgiram" com uma prioridade maior do que aquele que tem a mínima...quando ele terá acesso ao processador? Portanto, a alternativa A está correta e é o gabarito da questão.

Gabarito: Letra A

6. **(CESGRANRIO - 2018 - LIQUIGÁS - Profissional Júnior - Arquiteto de Soluções)**

A gerência do processador visa a otimizar o seu uso a partir do emprego de técnicas de escalonamento de processos. Dentre os critérios adotados para interromper o processo que está em execução, o término da fatia de tempo é amplamente utilizado pelos sistemas operacionais. Esse critério é adotado no escalonamento
A) Shortest Job First

B) First In First Out
C) First Come First Served
D) Round Robin (Circular)
E) por Prioridade

Comentários:

No Round Robin, cada processo na fila de prontos recebe uma pequena fatia de tempo para executar na CPU, e após esse tempo, é preterido para o próximo processo na fila. Essa abordagem é diferente da adotada pelos outros algoritmos listados, como Shortest Job First e First Come First Served, que não se baseiam em fatias de tempo fixas para a execução dos processos. O escalonamento por Prioridade também não necessariamente utiliza fatias de tempo para decidir a troca de processos. O First In First Out é apenas outra denominação para o First Come First Served, que também não usa fatias de tempo. Portanto, a característica de interrupção após uma fatia de tempo definida é específica do escalonamento Round Robin.

Gabarito: D

7. **(CESGRANRIO - 2018 - Transpetro - Engenheiro Júnior - Automação)**

Em sistemas operacionais multitarefas e interativos é comum utilizar um algoritmo de escalonamento de processos. Um algoritmo amplamente usado é conhecido como round-robin. Nesse algoritmo, cada processo

A) vai para uma fila de acordo com sua prioridade. As filas de maior prioridade são executadas primeiro. Cada fila é executada em ordem de chegada. Quando todas as filas são executadas, inicia-se de novo pela fila de maior prioridade.
B) recebe um número. A CPU seleciona aleatoriamente um dos processos em espera e o executa por um tempo fixo. Se esse tempo se esgota, a CPU seleciona aleatoriamente outro processo para substituí-lo.
C) possui um grau de importância que define sua ordem em uma lista de espera. Na sua vez, o processo é executado por um tempo fixo. Se esse tempo é ultrapassado, a CPU dá o controle para o próximo da lista, e o processo que estava sendo executado volta para uma posição à frente de todos os outros processos de menor importância
D) na lista de espera recebe um intervalo de tempo fixo em que é permitido executar. Na sua vez, se esse intervalo é ultrapassado, a CPU dá o controle para o próximo processo da lista, e o processo que estava sendo executado vai para o fim da lista.
E) na fila de espera possui uma prioridade. As prioridades são usadas para calcular o intervalo de tempo que o processo deve ficar na CPU, quanto maior a prioridade, maior o tempo. Quando

esse tempo se esgota, a CPU dá o controle para o próximo processo da lista, e o processo que estava sendo executado vai para o fim da lista.

Comentários:

A alternativa correta é D, pois no algoritmo de escalonamento Round Robin, cada processo na fila de espera recebe um intervalo de tempo fixo (conhecido como quantum) para executar na CPU. Quando este intervalo de tempo se esgota, independente do processo ter concluído sua tarefa ou não, a CPU passa o controle para o próximo processo na fila. O processo que estava executando é então movido para o final da fila de espera. Este método cria uma rotatividade justa e previsível entre todos os processos, garantindo que cada um receba uma quantidade igual de tempo de CPU, sem favorecer nenhum processo baseado em prioridade ou outro critério. As outras alternativas descrevem diferentes métodos de escalonamento que não correspondem ao funcionamento do Round Robin.

Gabarito: D

8. **(CESGRANRIO - 2018 - Transpetro - Analista de Sistemas Júnior - Infraestrutura)**
A política de escalonamento estabelece os critérios utilizados para selecionar o processo que fará uso do processador.

No escalonamento não preemptivo, quando um processo está em execução,

A) apenas o sistema operacional pode ocasionar a perda do uso do processador.
B) qualquer processo em espera pode ocasionar a perda do uso do processador.
C) qualquer processo pronto pode ocasionar a perda do uso do processador.
D) nenhum evento externo pode ocasionar a perda do uso do processador.
E) nem mesmo o próprio processo pode ocasionar a perda do uso do processador.

Comentários:

A alternativa correta é D, pois no escalonamento não preemptivo, uma vez que um processo começa a ser executado, ele mantém o controle da CPU até que ele mesmo libere a CPU, seja concluindo sua execução ou realizando uma operação de entrada/saída que o bloqueia. Em outras palavras, em um escalonamento não preemptivo, nenhum evento externo, como a intervenção de outro processo ou do sistema operacional, pode causar a preempção do processo em execução. Este tipo de escalonamento contrasta com o escalonamento preemptivo, no qual o sistema operacional pode interromper um processo em execução para ceder a CPU a outro

processo. As outras alternativas descrevem cenários que seriam aplicáveis ao escalonamento preemptivo ou situações que não são típicas de qualquer forma de escalonamento de processos.

Gabarito: D

QUESTIONÁRIO DE REVISÃO E APERFEIÇOAMENTO

A ideia do questionário é elevar o nível da sua compreensão no assunto e, ao mesmo tempo, proporcionar uma outra forma de revisão de pontos importantes do conteúdo, a partir de perguntas que exigem respostas subjetivas.

São questões um pouco mais desafiadoras, porque a redação de seu enunciado não ajuda na sua resolução, como ocorre nas clássicas questões objetivas.

O objetivo é que você realize uma auto explicação mental de alguns pontos do conteúdo, para consolidar melhor o que aprendeu ;)

Além disso, as questões objetivas, em regra, abordam pontos isolados de um dado assunto. Assim, ao resolver várias questões objetivas, o candidato acaba memorizando pontos isolados do conteúdo, mas muitas vezes acaba não entendendo como esses pontos se conectam.

Assim, no questionário, buscaremos trazer também situações que ajudem você a conectar melhor os diversos pontos do conteúdo, na medida do possível.

É importante frisar que não estamos adentrando em um nível de profundidade maior que o exigido na sua prova, mas apenas permitindo que você compreenda melhor o assunto de modo a facilitar a resolução de questões objetivas típicas de concursos, ok?

Nosso compromisso é proporcionar a você uma revisão de alto nível!

Vamos ao nosso questionário:

Perguntas

1. O que é um processo em sistemas operacionais e como ele difere de uma thread?

2. Descreva as quatro condições necessárias para a ocorrência de um deadlock.
3. O que é o efeito de convoy e em que tipo de escalonamento ele pode ocorrer?
4. Explique a diferença entre CPU Bound e IO Bound.
5. Como o livelock difere do deadlock?
6. O que é o estado de um processo zumbi?
7. Descreva o modelo de multithreading Many-to-One.
8. O que é um semáforo e como ele é usado na sincronização de processos?
9. O que é a política de escalonamento SJN e em que cenário ela é especialmente útil?
10. Como os sockets são usados em sistemas operacionais?
11. Explique o que é starvation e como ele pode ser prevenido.
12. O que é o Program Counter e qual é sua função?
13. Como o comando fork funciona em sistemas operacionais?
14. O que é um pipe e como ele facilita a comunicação entre processos?
15. Explique o modelo de multithreading One-to-One e suas vantagens e desvantagens.
16. Como o algoritmo de escalonamento Round Robin funciona?
17. O que é um monitor em termos de sincronização de processos?
18. Como um processo muda do estado pronto para o estado em execução?
19. O que é um processo IO Bound e como ele influencia o escalonamento?
20. Como o estado de um processo é preservado durante uma troca de contexto?

Perguntas e Respostas

1. O que é um processo em sistemas operacionais e como ele difere de uma thread?
   Resposta: Um processo é uma instância de um programa em execução com seu próprio espaço de endereçamento e recursos. Uma thread, por outro lado, é uma unidade menor de execução dentro de um processo e compartilha o espaço de endereçamento e recursos com outras threads no mesmo processo.

2. Descreva as quatro condições necessárias para a ocorrência de um deadlock.
   Resposta: As quatro condições são exclusão mútua, espera e posse (hold and wait), não preempção e espera circular.

3. O que é o efeito de convoy e em que tipo de escalonamento ele pode ocorrer?
   Resposta: O efeito de convoy ocorre quando processos mais curtos ficam bloqueados atrás de um processo longo, causando ineficiência. Ele pode ocorrer em escalonamento First-Come, First-Served (FCFS).

4. Explique a diferença entre CPU Bound e IO Bound.
   Resposta: CPU Bound refere-se a processos que gastam a maior parte do tempo executando cálculos, enquanto IO Bound refere-se a processos que gastam a maior parte do tempo em operações de entrada/saída.

5. Como o livelock difere do deadlock?
   Resposta: Enquanto o deadlock resulta em processos totalmente bloqueados sem progresso, o livelock ocorre quando os processos continuam ativos mas são incapazes de progredir na execução devido a um ciclo de tentativas falhas.

6. O que é o estado de um processo zumbi?
   Resposta: Um processo zumbi é um processo que terminou sua execução, mas ainda tem uma entrada na tabela de processos porque o processo pai ainda não leu seu status de saída.

7. Descreva o modelo de multithreading Many-to-One.
   Resposta: No modelo Many-to-One, várias threads de usuário são mapeadas para uma única thread de kernel, tornando o gerenciamento eficiente, mas podendo levar ao bloqueio global.

8. O que é um semáforo e como ele é usado na sincronização de processos?
   Resposta: Um semáforo é uma variável ou estrutura de dados usada para controlar o acesso a um recurso compartilhado, permitindo a cooperação entre processos ou threads.

9. O que é a política de escalonamento SJN e em que cenário ela é especialmente útil?
   Resposta: A política Shortest Job Next (SJN) atende ao processo com o menor tempo de execução estimado a seguir. É útil para minimizar o tempo médio de espera.

10. Como os sockets são usados em sistemas operacionais?
    Resposta: Os sockets são pontos de extremidade para comunicação entre dois computadores usando uma rede, permitindo a troca de dados entre processos em diferentes máquinas.

11. Explique o que é starvation e como ele pode ser prevenido.
    Resposta: Starvation ocorre quando um processo é continuamente negado acesso a um recurso. Pode ser prevenido através de algoritmos de escalonamento justos e mecanismos de envelhecimento.

12. O que é o Program Counter e qual é sua função?
    Resposta: O Program Counter é um registrador que contém o endereço da próxima instrução a ser executada pela CPU, orientando o fluxo de execução.

13. Como o comando fork funciona em sistemas operacionais?
    Resposta: O comando fork cria uma cópia exata do processo atual, resultando em um novo processo filho, com seu próprio espaço de endereçamento.

14. O que é um pipe e como ele facilita a comunicação entre processos?
    Resposta: Um pipe é um canal de comunicação que permite que processos relacionados troquem dados de maneira sequencial (FIFO), facilitando a interação entre eles.

15. Explique o modelo de multithreading One-to-One e suas vantagens e desvantagens.
    Resposta: No modelo One-to-One, cada thread de usuário é mapeada para uma thread de kernel. Oferece paralelismo real e isolamento de chamadas de sistema bloqueantes, mas tem um custo maior de criação e gerenciamento.

16. Como o algoritmo de escalonamento Round Robin funciona?
    Resposta: O Round Robin atende processos em turnos, dando a cada processo uma quantia fixa de tempo (quantum), permitindo uma distribuição justa do tempo de CPU.

17. O que é um monitor em termos de sincronização de processos?
    Resposta: Um monitor é uma abstração de sincronização que encapsula variáveis, condições e métodos, fornecendo controle de acesso mutualmente exclusivo a recursos compartilhados.

18. Como um processo muda do estado pronto para o estado em execução?
    Resposta: Um processo muda do estado pronto para o estado em execução quando é selecionado pelo escalonador e alocado à CPU para execução.

19. O que é um processo IO Bound e como ele influencia o escalonamento?
    Resposta: Um processo IO Bound passa a maior parte do tempo em operações de E/S. Esses processos frequentemente liberam a CPU, influenciando o escalonador a considerar a frequência de E/S no agendamento.

20. Como o estado de um processo é preservado durante uma troca de contexto?
    Resposta: O estado de um processo é preservado durante uma troca de contexto, salvando o contexto atual, incluindo registradores, program counter e informações de memória, para que possa ser restaurado quando o processo for retomado.

LISTA DE QUESTÕES ESTRATÉGICAS

1. **(FCC/Câmara Municipal-SP - 2014)** No escalonamento usando o algoritmo Round-Robin,

A) o escalonador seleciona o processo à espera com o menor tempo de execução estimado até a conclusão, reduzindo o tempo médio de espera, mas aumentando a variância dos tempos de resposta.

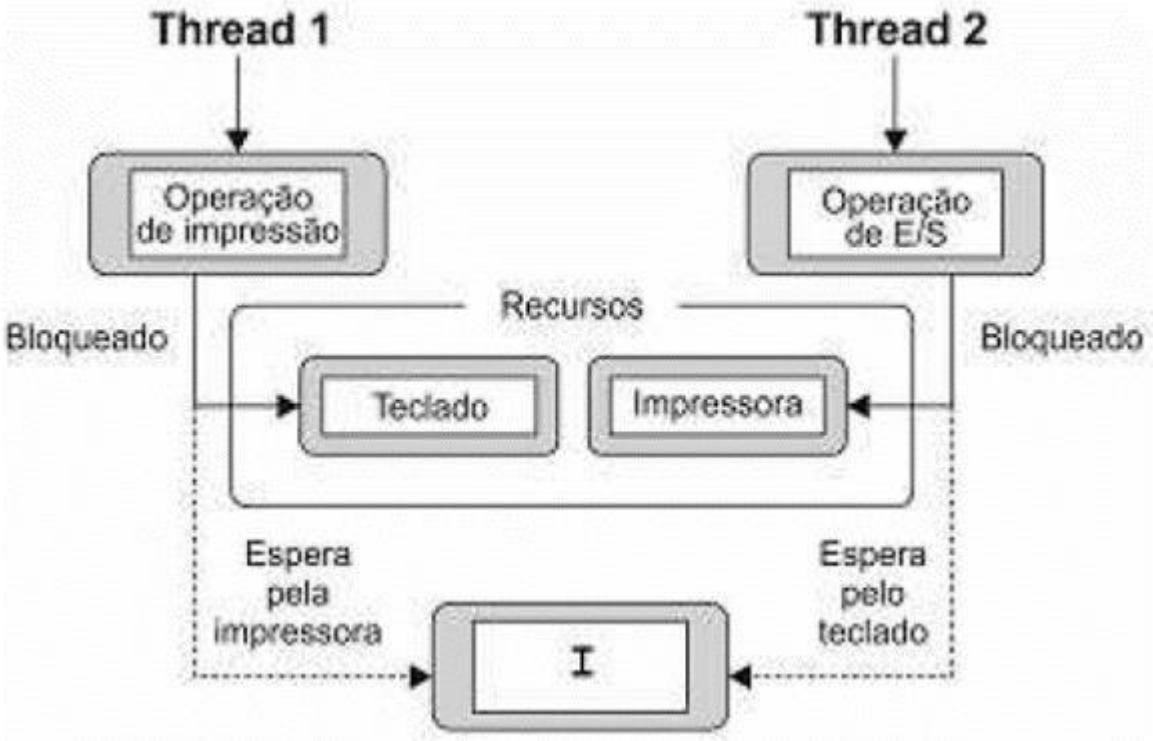
B) processos são despachados na ordem FIFO (First-in-First-Out), mas recebem uma quantidade limitada de tempo de processador denominada quantum.

C) a prioridade de cada processo é uma função não apenas do seu tempo de serviço, mas também do tempo que passou esperando pelo serviço.

D) o escalonador ajusta dinamicamente o comportamento do processo, de tal forma que o próximo processo a obter o processador seja aquele que chegar à frente da fila de nível mais alto, que não estiver vazia, na rede de filas.

E) o processo que tem o prazo de execução mais curto é favorecido, medindo a diferença entre o tempo que um processo requer para finalizar e o tempo restante até atingir o seu prazo final.

2. **(FCC/DPE-RS - 2017)** Considere a figura abaixo.



Do ponto de vista do sistema operacional, a situação indica que a caixa I deve ser preenchida com?

A) starvation.

B) multithreading.

C) superthreading.

D) deadlock.

E) hyperthreading.

3. **(FCC/DPE-RS - 2017)** Dentre as políticas de escalonamento de processos a seguir, a que apresenta maior probabilidade de ocasionar o starvation é a

A) Round Robin.
B) de tempo compartilhado.
C) First In First Out.
D) preemptiva.
E) não preemptiva.

4. **(IF-PA/IF-PA - 2019)** Em relação à gerência de processo, marque a alternativa CORRETA:

A) o processo é um programa em no estado de pronto.
B) os estados do processo são (execução, pronto, bloqueado ou espera).
C) a thread permite que apenas uma execução ocorra no mesmo ambiente do processo.
D) os sinais são mecanismos que permitem notificar o sistema operacional de eventos gerados pelo processador.
E) escalonamento é a escolha do processo, em estado de execução.

5. **(IESES/IGP-SC - 2017)** Acerca da gerência de processos dos sistemas operacionais, assinale a alternativa correta:

A) Um conjunto de processos está em estado de deadlock quando todos os processos no conjunto estão esperando por um evento que só pode ser causado por outro processo do conjunto.
B) Em um escalonamento preemptivo, um processo só perde o processador se terminar ou entrar em estado de espera.
C) No algoritmo de escalonamento de processos Round Robin, o escalonador sempre escolhe para execução o processo com menor expectativa de tempo de processamento. Esse algoritmo baseia-se no fato de que privilegiando processos pequenos o tempo médio de espera decresce.
D) Starvation é uma situação que não pode ocorrer quando um sistema operacional provê prioridades a processos.

6. **(CESGRANRIO - 2018 - LIQUIGÁS - Profissional Júnior - Arquiteto de Soluções)**

A gerência do processador visa a otimizar o seu uso a partir do emprego de técnicas de escalonamento de processos. Dentre os critérios adotados para interromper o processo que está em execução, o término da fatia de tempo é amplamente utilizado pelos sistemas operacionais. Esse critério é adotado no escalonamento

A) Shortest Job First
B) First In First Out
C) First Come First Served
D) Round Robin (Circular)
E) por Prioridade

7. **(CESGRANRIO - 2018 - Transpetro - Engenheiro Júnior - Automação)**

Em sistemas operacionais multitarefas e interativos é comum utilizar um algoritmo de escalonamento de processos. Um algoritmo amplamente usado é conhecido como round-robin. Nesse algoritmo, cada processo

A) vai para uma fila de acordo com sua prioridade. As filas de maior prioridade são executadas primeiro. Cada fila é executada em ordem de chegada. Quando todas as filas são executadas, inicia-se de novo pela fila de maior prioridade.
B) recebe um número. A CPU seleciona aleatoriamente um dos processos em espera e o executa por um tempo fixo. Se esse tempo se esgota, a CPU seleciona aleatoriamente outro processo para substituí-lo.
C) possui um grau de importância que define sua ordem em uma lista de espera. Na sua vez, o processo é executado por um tempo fixo. Se esse tempo é ultrapassado, a CPU dá o controle para o próximo da lista, e o processo que estava sendo executado volta para uma posição à frente de todos os outros processos de menor importância
D) na lista de espera recebe um intervalo de tempo fixo em que é permitido executar. Na sua vez, se esse intervalo é ultrapassado, a CPU dá o controle para o próximo processo da lista, e o processo que estava sendo executado vai para o fim da lista.
E) na fila de espera possui uma prioridade. As prioridades são usadas para calcular o intervalo de tempo que o processo deve ficar na CPU, quanto maior a prioridade, maior o tempo. Quando esse tempo se esgota, a CPU dá o controle para o próximo processo da lista, e o processo que estava sendo executado vai para o fim da lista.

8. **(CESGRANRIO - 2018 - Transpetro - Analista de Sistemas Júnior - Infraestrutura)**

A política de escalonamento estabelece os critérios utilizados para selecionar o processo que fará uso do processador.

No escalonamento não preemptivo, quando um processo está em execução,

A) apenas o sistema operacional pode ocasionar a perda do uso do processador.
B) qualquer processo em espera pode ocasionar a perda do uso do processador.
C) qualquer processo pronto pode ocasionar a perda do uso do processador.
D) nenhum evento externo pode ocasionar a perda do uso do processador.
E) nem mesmo o próprio processo pode ocasionar a perda do uso do processador.

Gabaritos

1. B
2. D
3. E
4. B
5. A
6. D
7. D
8. D

ESSA LEI TODO MUNDO CONHECE:
PIRATARIA É CRIME.

Mas é sempre bom revisar o porquê e como você pode ser prejudicado com essa prática.

1. 

Professor investe seu tempo para elaborar os cursos e o site os coloca à venda.

2. 

Pirata divulga ilicitamente (grupos de rateio), utilizando-se do anonimato, nomes falsos ou laranjas (geralmente o pirata se anuncia como formador de "grupos solidários" de rateio que não visam lucro).

3. 

Pirata cria alunos fake praticando falsidade ideológica, comprando cursos do site em nome de pessoas aleatórias (usando nome, CPF, endereço e telefone de terceiros sem autorização).

4. 

Pirata compra, muitas vezes, clonando cartões de crédito (por vezes o sistema anti-fraude não consegue identificar o golpe a tempo).

5. 

Pirata fere os Termos de Uso, adultera as aulas e retira a identificação dos arquivos PDF (justamente porque a atividade é ilegal e ele não quer que seus fakes sejam identificados).

6. 

Pirata revende as aulas protegidas por direitos autorais, praticando concorrência desleal e em flagrante desrespeito à Lei de Direitos Autorais (Lei 9.610/98).

7. 

Concurseiro(a) desinformado participa de rateio, achando que nada disso está acontecendo e esperando se tornar servidor público para exigir o cumprimento das leis.

8. 

O professor que elaborou o curso não ganha nada, o site não recebe nada, e a pessoa que praticou todos os ilícitos anteriores (pirata) fica com o lucro.

Deixando de lado esse mar de sujeira, aproveitamos para agradecer a todos que adquirem os cursos honestamente e permitem que o site continue existindo.