LCMAT – Laboratório de Ciências Matemáticas
Curso de Bacharelado em Ciência da Computação
Data: 27/ 09/2023 **Nota: ___/1,5**
Professor: Eduardo Carvalho
Grupo: 1
Integrantes: Gabriel Almeida, Enzo Marques, Gabriel Costa, Enzo Alberoni, Mariana Cossetti, Daniel Gomes

1 - Descreva a relevância dos processos tornando explícito o papel exercido por eles.

Os processos são fundamentais nos sistemas operacionais, permitindo a execução concorrente de tarefas, garantindo isolamento e proteção entre elas. Eles também são responsáveis pelo escalonamento de CPU, compartilhamento de recursos, independência entre aplicativos, tolerância a falhas e gestão eficaz dos recursos do sistema, como CPU e memória. Isso torna os processos cruciais para o funcionamento adequado e eficiente dos sistemas operacionais modernos.

2- Descreva a ideia de paralelismo associada a processos em dois contextos distintos: uma única CPU disponível; múltiplas CPUs disponíveis.

Uma única CPU disponível:

Nesse cenário, a CPU alterna rapidamente entre diferentes processos em execução, dando a impressão de que eles estão sendo executados simultaneamente. Isso é conhecido como multitarefa cooperativa ou multitarefa preemptiva. O sistema operacional aloca fatias de tempo para cada processo, permitindo que eles executem porções de suas instruções e, em seguida, alternem para o próximo processo. Isso é eficaz para manter a responsividade do sistema e melhorar a utilização da CPU.

Múltiplas CPUs disponíveis:

Quando há várias CPUs (ou núcleos) em um sistema, o paralelismo pode ser aproveitado de maneira mais eficaz. Cada CPU pode executar um processo separado de forma verdadeiramente simultânea, permitindo uma execução mais rápida e eficiente de tarefas. Isso é conhecido como processamento paralelo real e é amplamente utilizado em sistemas modernos para tarefas que podem ser divididas em subprocessos independentes, como renderização gráfica, simulações complexas, processamento de dados em lote e muito mais.

3 - Explique como um processo é criado.

Um processo pode ser criado de diversas formas, aqui estão alguns exemplos:

- **Inicialização do sistema:** Quando o Sistema operacional é inicializado, são criados processos em primeiro plano e em segundo plano. Sendo o primeiro associado com o usuário, e o segundo não associado com usuários.
- **Execução de uma chama de criação de processo por um processo em execução:** Novos processos podem ser criados, emitindo chamadas de sistema para criar um ou mais processos novos. Criar novos processos é útil quando o trabalho pode ser formulado como vários processos relacionados, interagindo de maneira independente
- **Solicitação do usuário:** O usuário pode começar um programa executando um comando, essas ações geram um processo e executa nele o programa selecionado.
- **Início de uma tarefa em lote:** É quando o sistema operacional decide que ele tem os recursos para executar outra tarefa, ele cria um novo processo e executa a próxima tarefa a partir de uma fila.

4 - Explique como ocorre a interação entre os processos, descrevendo seus estados e transições.

Um processo pode ter 3 estados:

- **Em execução:** Usando a CPU naquele momento
- **Pronto**: Executável, porém, temporariamente parado para deixar outro processo ser executado. Ou seja, não há uma nova CPU disponível para ele
- **Bloqueado**: Incapaz de ser executado até que haja uma entrada disponível.

E há 4 transições de estados:

- Uma chama de sistema bloqueia o processo em execução quando não há uma entrada disponível
- O escalonador de processos decide que o processo em andamento foi executado por tempo suficiente, e é momento de deixar outro processo ter seu tempo na CPU
- O escalonador de processo decide que todos os outros processos tiveram parcela justa de utilização da CPU e desta vez, está na hora do primeiro processo voltar a ser executado.
- Ocorre quando o evento externo pelo qual um processo estava esperando (como a chegada de alguma entrada) aconteça.

5 - Explique a diferença entre processos CPU-bound e I/O-bound.

CPU-BOUND (orientados à CPU) são processos que usam muitos recursos de CPU; seu tempo de execução é definido pelos ciclos de processador. Já o **I/O-bound** (orientado à E/S) utiliza muito do recursos de chamadas de sistema de entrada-saída. O tempo de execução é definido pela duração das operações de entrada e saída.

6 - O que são threads?

Threads são unidades de execução leves dentro de um processo que permitem a execução paralela de tarefas em um programa, compartilhando recursos e memória dentro do mesmo espaço de endereçamento do processo principal. Elas são usadas para melhorar o desempenho e a concorrência em sistemas de computador, permitindo que várias operações ocorram quase simultaneamente. Threads são mais eficientes que processos e são especialmente úteis em sistemas com várias CPUs.

7 - O que as threads associadas a um único processo compartilham?

As threads associadas a um único processo compartilham recursos essenciais, incluindo espaço de endereçamento, recursos do processo, código, dados, pilhas de chamadas e gerenciamento de memória. Isso permite a execução paralela de tarefas no mesmo processo.

8 - Descreva um exemplo prático do uso de múltiplas threads.

Suponha que você tenha uma coleção de imagens que precisam ser redimensionadas e aplicar filtros. Ao utilizar múltiplas threads, você pode dividir essa tarefa em várias partes e processar várias imagens ao mesmo tempo, melhorando significativamente o desempenho e a eficiência do aplicativo. Por exemplo, você pode criar uma thread para redimensionar imagens, outra para aplicar filtros e uma terceira para salvar as imagens processadas.

9 - Cite 1 vantagem e 1 desvantagem do uso das threads em relação ao uso de processos.

Vantagem das Threads em Relação aos Processos:

Eficiência de Recursos: Uma das principais vantagens das threads em relação aos processos é que as threads compartilham o mesmo espaço de endereço e recursos do processo pai. Isso significa que a criação e a troca de contexto de threads são geralmente mais eficientes em termos de uso de memória e recursos do que a criação e a troca de contexto de processos. As threads podem ser criadas e destruídas mais rapidamente, e a comunicação entre threads é mais eficiente, já que não envolve a sobrecarga de comunicação interprocessos.

Desvantagem das Threads em Relação aos Processos:

Falta de Isolamento: A principal desvantagem das threads em relação aos processos é a falta de isolamento. Como as threads compartilham o mesmo espaço de endereço e recursos do processo pai, um erro ou falha em uma thread pode afetar diretamente as outras threads do mesmo processo. Isso pode tornar o desenvolvimento e a depuração de programas com múltiplas threads mais complexos, pois os desenvolvedores precisam ser cuidadosos ao gerenciar o compartilhamento de recursos para evitar condições de corrida e outros problemas de concorrência. Em contraste, os processos oferecem um maior isolamento, tornando-os mais robustos em situações de falha de uma thread específica.

10 - Onde pode estar localizada a tabela de threads? Quais são as propriedades das threads?

A localização específica da tabela de threads pode variar dependendo do sistema operacional e de sua implementação. Em geral, a tabela de threads está localizada no kernel ou na unidade de gerenciamento de memória do sistema operacional.

Threads possuem diversas propriedades que definem seu comportamento e características:

- **ID do thread:** cada thread recebe um identificador exclusivo para distingui-lo de outros threads no sistema.
- **Estado do thread:** os threads podem estar em diferentes estados, como em execução, pronto, bloqueado ou encerrado.
- **Contador de programa:** rastreia a instrução atual que está sendo executada pelo thread.
- **Pilha:** Cada thread possui sua própria pilha, que é usada para armazenar variáveis locais e chamadas de função.
- **Conjunto de registradores:** Threads possuem seu próprio conjunto de registradores, incluindo contador de programa, ponteiro de pilha e registradores de uso geral.
- **Prioridade:** Threads podem ter prioridades diferentes, que determinam sua ordem de agendamento.
- **Dados específicos do thread:** os threads podem ter seus próprios dados que são acessíveis apenas para aquele thread específico.
- **Objetos de sincronização:** Threads podem usar objetos de sincronização como mutexes, semáforos e variáveis de condição para coordenar sua execução.
- **Contexto de execução:** Threads têm seu próprio contexto de execução, que inclui a pilha do thread, registros e outras informações relevantes.

- **Relacionamento pai-filho:** os threads podem ser organizados em um relacionamento pai-filho, onde um thread pai cria e gerencia um ou mais threads filhos.

11 - Descreva as chamadas Pthread_join e Pthread_yield.

A chamada Pthread_join bloqueia a execução do thread atual até que um thread especificado termine sua execução. Quando um thread termina, o valor de retorno é armazenado no local apontado por retval. Essa função é útil quando é necessário aguardar a conclusão de um tópico antes da obrigação com o restante do programa.

Já a chamada Pthread_yield é usada para ceder o processador especificamente para outro thread em um ambiente de programação multithread. Essa função permite que outros threads em execução tenham a oportunidade de serem escalonados e executados.

12 - Onde podem ser implementadas as threads e quais as vantagens e desvantagens entre esses modelos de execução?

Threads podem ser implementadas em diferentes níveis, incluindo nível de sistema operacional, nível de biblioteca e nível de aplicativo. As vantagens das threads nativas do sistema operacional incluem controle e eficiência, mas podem ser menos portáteis. Threads de biblioteca oferecem portabilidade, mas podem ser menos eficientes. Threads de aplicativo são criadas no próprio aplicativo, úteis para tarefas específicas, mas podem introduzir sobrecarga. A escolha depende dos requisitos do aplicativo, equilibrando concorrência e complexidade. Modelos híbridos também são comuns. Programação concorrente traz benefícios de desempenho, mas desafios de sincronização e condições de corrida. A seleção do modelo deve considerar eficiência, facilidade de desenvolvimento e manutenção. Em muitos casos, programação assíncrona e threads são combinadas para aproveitar o melhor de ambos os mundos.

13 - O quê são threads híbridas?

As threads híbridas são uma técnica que combina o uso de threads nativas e threads leves para otimizar o desempenho e a eficiência em programas concorrentes e paralelos, aproveitando as vantagens de ambas as abordagens, dependendo das necessidades específicas do aplicativo.

Algumas características das threads híbridas:

Threads Nativas (Pesadas): Essas threads são criadas e gerenciadas pelo sistema operacional. Elas são mais eficientes em termos de uso de recursos e são adequadas para operações intensivas de E/S e tarefas que podem ser executadas em paralelo em múltiplos núcleos de CPU. No entanto, a criação e a troca de contexto de threads nativas geralmente são mais caras em termos de tempo de execução.

Threads Leves (Green Threads): São threads gerenciadas pelo próprio programa, em vez de dependerem do sistema operacional. Elas são mais leves em termos de recursos e mais rápidas para criar e trocar de contexto do que as threads nativas. No entanto, geralmente não podem tirar pleno proveito de CPUs múltiplos, pois são executadas dentro de uma única thread nativa.

Utilização Estratégica: Threads híbridas podem ser usadas estrategicamente em um programa para otimizar o desempenho. Por exemplo, em um programa de servidor, você pode usar threads nativas para gerenciar conexões de rede de entrada e, em seguida, usar threads leves para processar solicitações individuais dentro de cada conexão. Isso permite que você atenda a várias conexões simultaneamente sem a sobrecarga de criar muitas threads nativas.

Balanceamento de Carga: Threads híbridas também podem ser usadas para melhorar o balanceamento de carga. Por exemplo, se você tiver um programa que processa um grande número de tarefas de maneira assíncrona, pode usar threads leves para distribuir essas tarefas entre um número menor de threads nativas, garantindo que a carga seja distribuída de maneira uniforme.

Escalabilidade: As threads híbridas podem ajudar a melhorar a escalabilidade de um aplicativo, permitindo que ele aproveite melhor os recursos de hardware disponíveis e, ao mesmo tempo, evite a sobrecarga de criação excessiva de threads nativas.