Desenvolvimento Mobile Profissional

App Dev 2023

Mark Joselli

mark.joselli@pucpr.br

Agenda

- Execução concorrente X Paralela
- Threads
  - Em android
  - Em iOS
- Otimização
  - Em android
  - Em iOS
- Exercicio

Cronograma

- ~~Aula 1 05/08/2023 – Introdução e TDD~~
- Aula 2 12/08/2023 – Threads e performance optimization
- Aula 3 02/09/2023 - Dependency Injection, Clean Code, SOLID
- Aula 4 16/09/2023 (EAD) - Design Patterns
- Aula 5 30/09/2023 - Arquitetura de Aplicativos Móveis
- Aula 6 14/10/2023 (EAD) - Projeto

Threads

Concorrente x Paralela

- A execução concorrente e paralela de tarefas são dois conceitos relacionados, mas distintos, na programação e na computação.
- Ambos se referem à ideia de realizar múltiplas tarefas ao mesmo tempo, mas diferem na forma como essas tarefas são executadas.

Execução Concorrente

- A execução concorrente envolve a realização de várias tarefas ao mesmo tempo, mas não necessariamente simultaneamente.
- As tarefas são alternadas rapidamente, dando a ilusão de que estão ocorrendo simultaneamente.
- Pode ser realizada usando uma única thread que alterna entre as tarefas ou usando múltiplas threads.
- É útil quando há tarefas interativas ou bloqueantes, como interações de usuário e operações de entrada/saída (I/O).
- Geralmente não aproveita totalmente a capacidade de processamento de múltiplos núcleos do processador.

Execução Paralela

- A execução paralela envolve a realização real de múltiplas tarefas ao mesmo tempo, simultaneamente, em processadores ou núcleos separados.
- Cada tarefa é executada de forma independente e em paralelo com as outras.
- Aproveita totalmente a capacidade de processamento de múltiplos núcleos do processador.
- É mais eficiente para tarefas que podem ser divididas em partes independentes e executadas simultaneamente.
- Geralmente é utilizado quando há necessidade de processamento intensivo, como cálculos complexos.

Execução Concorrente x Execução Paralela

- A execução concorrente é mais adequada quando há interações de usuário ou bloqueios envolvidos.
- A execução paralela é mais eficaz quando há recursos de hardware suficientes disponíveis para executar tarefas independentes simultaneamente.

Threads

- Threads são unidades independentes e concorrentes de execução em um programa de computador.
  - Cada thread representa um fluxo separado de instruções que pode ser executado simultaneamente com outras threads dentro do mesmo processo.
  - Essa abordagem permite que um programa realize múltiplas tarefas de forma paralela, melhorando a eficiência, a capacidade de resposta e a utilização dos recursos do sistema.
- As threads compartilham o mesmo espaço de endereçamento do processo pai, mas possuem seus próprios registradores, contador de programa e pilha de execução.

Threads

- Isso significa que cada thread tem sua própria sequência de instruções para executar, mas pode acessar as mesmas variáveis e recursos compartilhados pelo processo.
  - No entanto, o compartilhamento de recursos pode levar a problemas de concorrência, como condições de corrida e deadlocks.
- As threads podem ser usadas para realizar tarefas paralelas, como processamento intensivo, operações de entrada e saída (I/O), manipulação de interfaces de usuário e execução de tarefas assíncronas.
  - Elas são comumente usadas em sistemas operacionais, aplicativos de software e ambientes de programação para melhorar o desempenho e a capacidade de resposta, especialmente em sistemas com múltiplos núcleos de processamento.

Principais Características das Threads:

- Concorrência: As threads permitem a execução concorrente de tarefas, o que significa que várias threads podem estar ativas ao mesmo tempo, alternando entre a execução.
- Compartilhamento de Recursos: As threads compartilham o mesmo espaço de endereçamento e recursos do processo pai. Isso permite que elas acessem variáveis, dados e outras estruturas compartilhadas.
- Contexto Próprio: Cada thread tem seu próprio contexto de execução, incluindo contador de programa, registradores e pilha de chamadas. Isso permite que elas mantenham estados independentes.
- Comunicação: As threads podem se comunicar umas com as outras por meio de mecanismos de sincronização, como semáforos, mutexes e variáveis de condição.
- Escalonamento: O sistema operacional gerencia o escalonamento das threads, decidindo quando e por quanto tempo cada thread é executada. Isso garante uma alocação justa de recursos.

Aplicações das Threads:

- Melhoria da Responsividade da Interface do Usuário (UI): Threads são usadas para manter a interface do usuário ágil e responsiva, permitindo que operações de longa duração ocorram em segundo plano sem bloquear a interação do usuário.
- Processamento Paralelo: Threads são ideais para dividir tarefas intensivas em CPU em partes menores e executá-las simultaneamente em múltiplos núcleos de processamento, acelerando o processamento.
- Tarefas Assíncronas: Threads são usadas para executar tarefas assíncronas, como operações de E/S (entrada e saída), downloads, carregamento de recursos e processamento de dados, permitindo que o programa continue executando outras ações.

Aplicações das Threads:

- Concorrência em Redes: Threads permitem que aplicativos lidem com várias conexões de rede simultaneamente, permitindo a criação de servidores e clientes mais eficientes.
- Computação Distribuída: Em sistemas distribuídos, threads podem ser usadas para coordenar a execução de tarefas em diferentes máquinas, permitindo processamento paralelo em uma escala maior.
- Manipulação de Eventos: Threads podem ser usadas para manipular eventos em tempo real, como processamento de entradas de sensores, sinais ou interações de dispositivos.

Aplicações das Threads:

- Execução em Segundo Plano: Threads são usadas para executar tarefas em segundo plano, como sincronização de dados, atualizações de cache e processamento de logs, sem afetar o fluxo principal do aplicativo.
- Processamento Multimídia: Aplicações que lidam com áudio, vídeo e gráficos podem usar threads para decodificar, renderizar e processar esses tipos de dados de forma simultânea.
- Sistemas Operacionais: Threads são fundamentais para a criação de sistemas operacionais multitarefa, onde vários processos podem compartilhar recursos do sistema de maneira eficiente.
- Simulações e Modelagem: Threads podem ser usadas para criar simulações ou modelagens que executam várias partes de um sistema ao mesmo tempo, permitindo análises mais rápidas e abrangentes.

Threads em AppDev

- Responsividade da Interface do Usuário (UI): O uso de threads permite manter a interface do usuário responsiva, garantindo que as interações do usuário não sejam bloqueadas por tarefas demoradas, como carregamento de dados ou operações de rede.
- Operações de Rede: Threads podem ser usadas para realizar operações de rede em segundo plano, como requisições HTTP, downloads e uploads de arquivos, sem congelar a interface do usuário durante a espera pela resposta do servidor.
- Processamento de Dados Assíncrono: Threads permitem executar tarefas assíncronas em segundo plano, como processamento de dados, análises e transformações, sem interferir no fluxo principal do aplicativo.

Threads em AppDev

- Atualizações de Dados em Tempo Real: Threads podem ser usadas para atualizar dados em tempo real, como feeds de redes sociais ou dados de sensores, garantindo que as informações sejam exibidas assim que estiverem disponíveis.
- Processamento de Imagens e Vídeos: Threads podem ser usadas para processar imagens, aplicar filtros e realizar outras manipulações gráficas sem afetar a resposta da interface do usuário.
- Execução de Tarefas de Longa Duração: Threads são ideais para executar tarefas que podem levar algum tempo para serem concluídas, como conversões de formato de arquivo, processamento de áudio/vídeo e renderização.

Threads em AppDev

- Sincronização de Dados: Threads podem ser usadas para sincronizar dados locais com servidores remotos, garantindo que as informações estejam atualizadas e consistentes.
- Manipulação de Múltiplos Dispositivos: Threads podem ser usadas para lidar com a interação de vários dispositivos ou eventos, como jogos multiplayer ou aplicativos IoT (Internet das Coisas).
- Processamento de Eventos Assíncronos: Threads permitem a execução de código em resposta a eventos assíncronos, como notificações de sistema ou eventos do usuário.
- Execução em Segundo Plano: Threads podem ser usadas para executar tarefas em segundo plano, como backup de dados, envio de notificações e atualizações silenciosas, sem interromper o fluxo principal do aplicativo.

Threads no Android

- No Android, existem dois tipos principais de threads que são frequentemente usados para executar tarefas concorrentes:
  - a UI Thread (também conhecida como Main Thread)
  - e as Background Threads (Threads em segundo plano).

Threads no Android

- UI Thread (Main Thread):
  - A UI Thread é a thread principal do aplicativo, responsável por manipular a interface do usuário e responder a eventos de entrada, como toques na tela.
  - Qualquer operação que afete a interface do usuário, como atualizações de widgets ou exibição de diálogos, deve ser feita na UI Thread.
  - É importante manter a UI Thread responsiva para evitar que o aplicativo pareça lento ou não responda (ANR - Application Not Responding).
  - No entanto, operações demoradas ou bloqueantes não devem ser executadas na UI Thread, pois podem causar atrasos na interface do usuário.

Threads no Android

- Background Threads (Threads em Segundo Plano):
  - Background Threads são threads adicionais que podem ser criadas para executar tarefas demoradas ou bloqueantes sem afetar a UI Thread.
  - Eles são usados para realizar operações em segundo plano, como operações de E/S, cálculos intensivos e tarefas de rede.
  - O uso de Background Threads mantém a UI Thread responsiva, garantindo uma experiência de usuário mais suave.
  - Background Threads não devem acessar diretamente elementos da interface do usuário, pois isso pode causar problemas de sincronização.
  - Exemplos de Background Threads incluem Threads tradicionais, AsyncTask, Executors, HandlerThread, e bibliotecas como RxJava e Kotlin Coroutines.

Threads no Android

- AsyncTask: Uma classe que simplifica a criação de Background Threads e lida com a comunicação entre a UI Thread e a thread em segundo plano. É útil para tarefas curtas e não intensivas em CPU.
- HandlerThread: Uma classe que estende a funcionalidade de Thread para fornecer uma Looper que pode ser usada para postar mensagens para uma thread em segundo plano. É útil para operações que requerem um loop de mensagem, como atualizações periódicas.
- RxJava: Uma biblioteca reativa que permite compor operações assíncronas de maneira mais concisa e legível. É amplamente usado para manipulação assíncrona de dados e eventos.
- Kotlin Coroutines: Uma abstração para programação assíncrona que simplifica a escrita de código assíncrono, tornando-o mais sequencial e legível. É uma maneira moderna de lidar com threads e tarefas assíncronas.
- Cada tipo de thread tem seus próprios cenários de uso e considerações. A escolha do tipo de thread a ser usado dependerá das necessidades específicas do aplicativo e das tarefas a serem executadas.

Threads no iOS

- No iOS, a plataforma de desenvolvimento da Apple, também existem vários tipos de threads que podem ser usados para executar tarefas concorrentes.
- Aqui estão os principais tipos de threads no iOS:
  - Main Thread (Thread Principal):
  - Assim como no Android, o Main Thread no iOS é responsável por manipular a interface do usuário e responder a eventos.
  - Todas as atualizações de interface do usuário devem ser feitas no Main Thread para garantir a responsividade e a consistência da UI.
  - Tarefas demoradas ou bloqueantes não devem ser executadas no Main Thread, pois isso pode levar a uma experiência do usuário ruim.

Threads no iOS

- Background Threads (Threads em segundo plano):
  - Assim como no Android, é importante executar tarefas demoradas ou bloqueantes em threads separadas para manter a UI responsiva.
  - No iOS, uma abordagem comum para criar threads em segundo plano é usar a classe `DispatchQueue`, que permite a criação e gerenciamento de threads concorrentes.
  - Você pode usar a `DispatchQueue` para agendar blocos de código para execução em threads em segundo plano, gerenciando automaticamente o escalonamento.

Threads no iOS

- Global Dispatch Queues:
  - O Grand Central Dispatch (GCD) fornece Global Dispatch Queues, que são filas pré-configuradas que executam tarefas em threads em segundo plano.
  - Existem diferentes níveis de prioridade para essas filas, como `background`, `default` e `high`. Você pode escolher a fila apropriada com base nas necessidades da tarefa.

Threads no iOS

- Private Dispatch Queues:
  - Além das filas globais, você pode criar suas próprias Private Dispatch Queues usando o GCD. Isso oferece mais controle sobre a execução das tarefas.
  - As Private Dispatch Queues são úteis para tarefas específicas, como operações de longa duração ou tarefas que exigem sincronização precisa.

Threads no iOS

- Operations:
  - A classe `Operation` e a classe `OperationQueue` oferecem uma abstração mais alta em relação às threads e são usadas para executar tarefas assíncronas de maneira mais estruturada.
  - As operações podem ser usadas para executar tarefas complexas, gerenciando automaticamente as dependências entre elas e permitindo o controle mais avançado sobre a execução.

Threads no iOS

- Threads Tradicionais (pthread):
  - O iOS também oferece suporte a threads POSIX, conhecidas como pthreads.
  - Essas threads oferecem mais controle e flexibilidade, mas geralmente são usadas em cenários específicos.

Otimização

Otimização

- A otimização de desempenho em aplicativos móveis é de suma importância devido ao impacto direto que tem na experiência do usuário e no sucesso geral do aplicativo.
- Algumas razões pelas quais a otimização de desempenho é crucial:
  - Experiência do Usuário Melhorada: Aplicativos móveis com desempenho otimizado oferecem respostas mais rápidas, transições suaves e tempos de carregamento reduzidos.
    - Isso resulta em uma experiência do usuário mais agradável e satisfatória, o que é essencial para reter os usuários e conquistar avaliações positivas na loja de aplicativos.
  - Engajamento e Retenção: Um aplicativo que responde rapidamente e não apresenta travamentos é mais propenso a manter os usuários engajados e a incentivá-los a usar o aplicativo com frequência.
    - Usuários são mais propensos a abandonar aplicativos que apresentam atrasos frequentes ou problemas de desempenho.

Otimização

- Avaliações e Reputação: Aplicativos que oferecem um excelente desempenho tendem a receber avaliações positivas dos usuários.
  - Avaliações positivas têm um impacto direto na classificação e na visibilidade do aplicativo nas lojas, aumentando sua chance de ser descoberto por novos usuários.
- Consumo de Recursos do Dispositivo: Aplicativos que não são otimizados podem consumir mais recursos do dispositivo, como CPU, memória e bateria.
  - Isso pode afetar a duração da bateria do dispositivo, além de causar aquecimento excessivo e possíveis problemas de estabilidade.

Otimização

- Diversidade de Dispositivos: Com a variedade de dispositivos móveis disponíveis, cada um com diferentes especificações de hardware, é fundamental otimizar o desempenho para garantir uma experiência consistente em todos os dispositivos, desde os mais antigos até os mais recentes.
- Concorrência no Mercado: O mercado de aplicativos móveis é altamente competitivo.
  - Aplicativos que oferecem um desempenho superior têm uma vantagem competitiva sobre os concorrentes, atraindo mais usuários e mantendo-se à frente.
- Crescimento da Base de Usuários: Quando os usuários experimentam um aplicativo rápido e responsivo, eles têm mais probabilidade de compartilhá-lo com outras pessoas, o que pode levar a um crescimento orgânico da base de usuários.

Perfis de Desempenho

- Diferentes aspectos de desempenho:
  - como velocidade,
  - consumo de memória,
  - consumo de bateria
  - e latência.

Velocidade

- Refere-se à rapidez com que um aplicativo responde às interações do usuário e executa tarefas.
- Um aplicativo rápido oferece transições suaves entre telas, carregamento instantâneo de conteúdo e resposta imediata a toques e gestos.
- A otimização de velocidade envolve minimizar atrasos e gargalos no código, otimizar consultas de banco de dados e reduzir o tempo de inicialização do aplicativo.

Consumo de Memória

- Refere-se à quantidade de memória RAM utilizada pelo aplicativo durante sua execução.
- Um alto consumo de memória pode levar a travamentos, fechamento repentino do aplicativo e afetar o desempenho de outros aplicativos em execução.
- A otimização de consumo de memória envolve a liberação adequada de recursos não utilizados, o uso eficiente de objetos e a minimização de vazamentos de memória.

Consumo de Bateria

- Refere-se à quantidade de energia consumida pelo aplicativo durante o uso.
- Aplicativos que consomem muita bateria podem levar a uma experiência insatisfatória para o usuário e reduzir a duração da bateria do dispositivo.
- A otimização de consumo de bateria envolve a redução do uso de CPU e GPU, o gerenciamento eficiente de atualizações em segundo plano e a implementação de estratégias de economia de energia.

Latência

- Refere-se ao tempo de resposta entre a interação do usuário e a ação correspondente do aplicativo.
- Uma alta latência pode levar a uma sensação de lentidão e frustração por parte do usuário.
- A otimização da latência envolve minimizar o tempo de processamento de solicitações, reduzir a latência de rede e otimizar a renderização gráfica.

Android Otimization

- Uso de ferramentas de análise, como o Android Profiler, para identificar gargalos de desempenho e áreas problemáticas
- Evite vazamentos de memória, liberando referências não utilizadas. Use ferramentas como o LeakCanary para identificar vazamentos.
  - https://square.github.io/leakcanary/
- Armazene em cache dados frequentemente usados para evitar chamadas excessivas ao servidor. Use bibliotecas como o Glide ou Picasso para gerenciar o cache de imagens.
- Escolha bibliotecas bem otimizadas e bem mantidas para funcionalidades específicas, como comunicação em rede, injeção de dependência, etc.
- Utilize ferramentas como o Android Layout Inspector para identificar layouts complexos ou ineficientes que podem causar sobrecarga de renderização.
- Utilize formatos de imagem vetorizados sempre que possível ou reduza o tamanho das imagens usando ferramentas de compressão.

iOS Otimization

- Use o Instruments, uma ferramenta do Xcode, para realizar perfis de desempenho e identificar gargalos, como alocação excessiva de memória ou tempo de CPU elevado.
- Carregue dados conforme necessário para evitar carregamentos excessivos e lentidão na inicialização. Isso é especialmente importante para tabelas e coleções.
- Utilize cache para armazenar dados frequentemente usados e evite fazer requisições desnecessárias ao servidor.
- Utilize formatos de imagem otimizados, como o PNG otimizado (PNGcrush) ou o formato WebP.
- Utilize bibliotecas bem otimizadas e atualizadas, verificando se elas não têm impacto negativo no desempenho.

Desafio TDD

- Criar um aplicativo que faça o download de várias imagens (5) em paralelo usando threads separadas e exiba-as na interface do usuário.
  - https://picsum.photos/