

**UNIVERSIDADE ANHANGUERA – CAMPUS PASSO FUNDO**

**CURSO – ANALISE E DESENVOLVIMENTO DE SISTEMAS – MODALIDADE A DISTANCIA.**

**ALINE DOS REIS CONRADO**

**PASSO FUNDO – RS**

**2022**

**ALINE DOS REIS CONRADO**

**PORTFOLIO INTERDISCIPLINAR INDIVIDUAL.**

**ADS – BackEnd – Web – Mobile – DevOps.**

**Orientador (a): Joice Siqueira Lima**

**PASSO FUNDO – RIO GRANDE DO SUL**

**2022**

**SUMARIO.**

**1** INTRODUÇÃO.

**2.1** TAREFA 1.

**2.2** TAREFA 2.

**2.3** TAREFA 3.

**2.4** TAREFA.

**3** CONCLUSÃO.

**4** REFERENCIAS.

**1** **INTRODUÇÃO**

O desenvolvimento da produção textual, tem como finalidade a apresentação de conceitos e fundamentos sobre diversos aspectos e disciplinas; buscando evidenciar características sobre assuntos pertinentes em áreas específicas de atuação. Este projeto será desenvolvido discorrendo de assuntos concernentes as disciplinas estudadas no decorrer do semestre acerca de Computação em Nuvem, Sistemas Operacionais, Programação web, Optativa 2, e Desenvolvimento Mobile.

**Tarefa 1:** Vamos supor que você vai iniciar uma jornada empreendedora. Junto com alguns sócios, decidiu criar um negócio a partir da ideia de uma aplicação inovadora. Esta aplicação precisa de um ambiente de TI com recursos computacionais de alto desempenho e ambiente de nuvem devido à redução de cursos e abstração da complexidade da infraestrutura. O próximo passo é determinar qual dos modelos de serviço é mais apropriado para este caso: SaaS, PaaS ou IaaS. Descreve analisando caso a caso.

RESPOSTA:

**Infraestructure as a Service** (IaaS).

Com isso, é possível **terceirizar** toda **infraestrutura da aplicação** e se preocupar apenas com o **desenvolvimento da aplicação**.

Descrição A nomenclatura ****IaaS, PaaS e SaaS**** refere-se ao modelo de ****fornecimento de soluções em nuvem****.

Os termos significam respectivamente:

* ****IaaS:**** Infraestrutura como Serviço.
* ****PaaS:**** Plataforma como Serviço.
* ****SaaS:**** Software como Serviço.

**Tarefa 2:** Com o desenvolvimento dos sistemas, surgiu a necessidade de, dentro de um mesmo processo, possuir trechos que executassem concorrentemente, e aí temos o conceito de thread.

a) Definição de thread;

RESPOSTA:

Thread (um termo em inglês para **Linha ou Encadeamento de Execução**) é um programa que trabalha com um subsistema , sendo uma forma de um processo se autodividir em duas ou mais tarefas. Essas tarefas múltiplas podem ser executadas simultaneamente para rodar mais rápido do que um programa em um único bloco ou praticamente juntas, mas que conseguem ser tao rápidas que da a impressão que estão trabalhando em conjunto ao mesmo tempo.

b) Pesquise os estados das threads;

RESPOSTA:

**(1) Ready (2) Running (3) Waiting (4) Delayed (5) Blocked**

* Quando um aplicativo deve ser processado, ele cria um encadeamento.
* Em seguida, são alocados os recursos necessários (como uma rede) e entram na fila ****PRONTO.****
* Quando o agendador de thread (como um agendador de processo) atribui o thread com o processador, ele entra na fila ****RUNNING****.
* Quando o processo precisa que algum outro evento seja disparado, que está fora de seu controle (como outro processo a ser concluído), ele faz a transição da fila ****RUNNING**** para ****WAITING****.
* Quando o aplicativo tem a capacidade de atrasar o processamento do encadeamento, quando necessário pode atrasar o encadeamento e colocá-lo em hibernação por um período específico de tempo. O thread então faz a transição da fila ****RUNNING**** para ****DELAYED****. Um exemplo de atraso de thread é a soneca de um alarme. Depois de tocar pela primeira vez e não for desligado pelo usuário, ele tocará novamente após um determinado período de tempo. Durante esse tempo, o thread é colocado no modo de espera.
* Quando o thread gera uma solicitação de I / O e não pode se mover mais até que seja concluído, ele faz a transição da fila ****RUNNING**** para ****BLOCKED****.
* Após a conclusão do processo, o encadeamento passa de ****RUNNING**** para ****FINISHED****.

c) Os benefícios da programação multithread;

RESPOSTA:

* **Capacidade de Resposta**: O multithreading em um app interativo pode permitir que um programa continue em execução mesmo se uma parte dele estiver bloqueada ou executando uma operação que leva mais tempo, aumentando a capacidade de resposta ao usuário.
* **Compartilhamento de recursos**: Os processos podem compartilhar recursos apenas por meio de técnicas como Passagem de mensagens e Memória compartilhada. Técnicas das quais devem ser organizados pelo programador. A vantagem de compartilhar códigos e dados e que permite que um app tenha vários threads de atividade no mesmo espaço de endereço.
* **Economia**: Alocar memória e recursos para a criação de processos e um trabalho caro em termos de tempo e espaço. Visto que os encadeamentos compartilham memória com o processo ao qual pertence é mais econômico criar e alternar encadeamentos de contexto.
* **Ecabilidade**: Os benefícios da multiprogramação aumentam significantemente no caso de arquitetura de multiprocessador onde os threads podem ser executados paralelamente em vários processadores. Se houver apenas um thread não será possível dividir os processos em tarefas menores que diferentes processadores podem executar. O processo de thread único pode ser executado apenas em um processador independente de quantos processadores estão disponíveis.

d) Operações de thread

RESPOSTA:

Há quatro operações básicas na gestão de threads: criar, terminar, thread join e thread yield

* Criar (*thread creation*)

Basicamente uma thread pode criar outra(s), sendo que depois essas mesmas threads são executas ‘simultaneamente’. A thread criadora é a thread-mãe e a thread criada é a thread-filho. Threads incluídas na *função main* quando executadas podem criar threads-filho. No diagrama a seguir há a thread A que executa inicialmente. Mais tarde é criada a thread B indicada no ponto amarelo. Depois de criadas, a thread A e thread B executam simultaneamente. Em seguida a thread A pode criar uma ou mais threads (por exemplo uma thread C). Depois de criada a thread C, há três threads executando simultaneamente e todas disputam o uso da CPU. Entretanto, a thread que pode ser executada a qualquer momento não é de conhecimento da CPU.

* Terminar (*thread termination*)

Para maioria dos casos, as threads não são criadas e executadas eternamente. Depois de terminado o seu objectivo, a thread termina. No facto, a thread que criou estas duas threads-filho termina também, porque sua tarefa atribuída se completa. Na matrix de multiplicação (*matrix multiplication*), uma vez que o valor de C[i,j] é computado, a thread correspondente termina. Em geral quando a tarefa atribuída a thread completa, a thread pode ser terminada. Além disso, se a thread-mãe terminar, todas as threads filho terminam também. Porque isso é importante? Isso é importante porque as threads-filho compartilham recursos com a thread-mãe, incluindo variáveis. Quando a thread-mãe termina, todas as variáveis são perdidas e a thread-filho não poderá aceder aos recursos que a thread-mãe possuía. Assim, se a thread-mãe terminar mais cedo que a thread-filho haverá um problema!

* Sincronizar(*Thread Join*)

Imagine a seguinte situação: Você está estudando para uma prova. Então você pede o seu irmão mais novo para comprar uma pizza. Neste caso você é a thread principal e seu irmão a thread-filho. Uma vez que você deu a ordem, você e seu irmão começam a “executar uma tarefa” simultaneamente. Agora há dois casos a se considerar: Primeiro: Seu irmão traz a pizza e termina enquanto você estuda. Nesse caso você pode parar de estudar e comer a pizza. Segundo: Você acaba de estudar mais cedo e dorme e depois a pizza chegará.

A junção de threads (thread join) é destinada para resolver este problema. A thread pode executar o *thread join* e aguardar até a outra thread terminar. No caso acima, você é a thread principal (*thread main*) e deve executar o *thread join*aguardando o seu irmão (thread-filho) terminar. Em geral o *thread join* é utilizado para a thread-mãe se sincronizar com uma das threads-filho.

* Thread Yield (Rendimento da thread)

Suponha que você executa um certo número de programas o tempo todo no computador. Isso é possível devido a CPU escalonar pouco a pouco outros ciclos da CPU, assim outros programas podem ser executados. Isso pode ser um problema de política de planeamento do Sistema Operativo. Entretanto, quando escrevemos programas com múltiplas threads, temos que fazer correctamente para que algumas threads não ocupem a CPU eternamente, ou por um tempo muito longo sem abandoná-lo. Senão terminar na situação acima quando uma ou duas threads executam enquanto outras simplesmente esperam para retornar. Liberamos espaço na memória graças a *thread yield*. Quando a thread executa o *thread yield*, a execução da thread é suspensa e a CPU passa para uma outra thread em execução. Essa thread aguardará até a CPU tornar-se disponível novamente.

Tarefa 3: Descreva detalhadamente todos os métodos HTTP existentes e cite um exemplo de utilização para cada um deles.

RESPOSTA:

* **GET:** Essa é a requisição mais comum de todas. Através dessa requisição nós pedimos a representação de um recurso: que pode ser um arquivo html, xml, json, etc.
* **POST:** O método POST é utilizado quando queremos ****criar**** um recurso. Quando usamos POST, os dados vão no corpo da requisição e não na URI.
* **DELETE:** Exclui o recurso especificado.
* **TRACE:** Devolve a mesma requisição que for enviada veja se houve mudança e/ou adições feitas por servidores intermediários.
* **OPTIONS:** Retorna os métodos HTTP suportados pelo servidor para a URL especificada.
* **PATCH:** Serve para atualizar ****partes**** de um recurso, e não o recurso todo.
* **CONNECT:** Converte a requisição de conexão para um túnel TCP/IP transparente, geralmente para facilitar a comunicação criptografada com SSL (HTTPS) através de um proxy HTTP não criptografado.
* **HEAD:** Retorna somente os cabeçalhos de uma resposta.
* **Idempotência:** Os métodos que não alteram nada no servidor e que podemos chamar várias vezes são o que chamamos de métodos ****idempotentes****.

GET, OPTIONS, HEAD, PUT, TRACE, CONNECT e DELETE são idempotentes.

**Tarefa 4:** Em aplicações Android podemos trabalhar com a persistência de dados locais por meio da criação de bancos de dados utilizando a biblioteca SQLite. Descreva as principais características do SQLite para aplicações Android, bem como as principais vantagens e desvantagens de sua utilização.

RESPOSTA:

**SQLite é um dos bancos de dados relacional mais conhecidos, por ser capaz de criar uma estrutura com um banco de dados embutido.** O SQLite é um mini-SGBD, capaz de criar um arquivo, ler e escrever diretamente nele. O SQLite já está disponível na plataforma Android, não necessitando de instalação, configuração ou administração, suporta commit e rollback, além de ser um banco de dados gratuito.

**Vantagens:**

É estável, multiplataforma e compatível com versões anteriores;

* Seu código é de domínio público e gratuito, ou seja, todos podem utilizar;
* Não necessita de instalação ou configuração;
* Guarda o banco de dados em um único arquivo.

**Desvantagens:**

* É mais simples e não suporta um volume grande de informações;
* É mais restrito em relação aos formatos dos arquivos aceitos.

**CONCLUSÃO.**

Este Projeto Integrado propiciou a oportunidade de aplicação prática dos conceitos teóricos estudos nas disciplinas ao decorrer do semestre, desenvolvendo no aluno o conhecimento necessário para o futuro cotidiano profissional, e uma vivência prática dentro das organizações. A elaboração deste projeto, trouxe um bom aprendizado e uma possibilidade de potencializar os estudos e os conhecimentos sobre as disciplinas norteadoras do semestre.

**REFERENCIAS.**

https://www.techtudo.com.br/noticias/2019/01/o-que-sao-threads-e-para-que-servem-em-um-processador.ghtml

https://rockcontent.com/br/blog/sqlite/

https://brasil.softlinegroup.com/sobre-a-empresa/blog/iaas-paas-saas-nuvem