INTERNET

COMO A INTERNET FUNCIONA

A Internet é uma rede global de computadores interconectados que permite a troca de informações e dados em escala mundial. Ela funciona com base em uma arquitetura descentralizada e é construída sobre uma série de protocolos e tecnologias que permitem a comunicação entre dispositivos em diferentes locais. Aqui estão os principais componentes e conceitos de como a Internet funciona:

- Dispositivos de Acesso:
  Os dispositivos de acesso são os computadores, smartphones, tablets e outros dispositivos que as pessoas usam para se conectar à Internet. Eles se conectam à Internet através de provedores de serviços de Internet (ISP).

- Protocolo IP:
  O Protocolo da Internet (IP) é o protocolo central que permite a identificação e o encaminhamento de pacotes de dados na rede. Cada dispositivo conectado à Internet possui um endereço IP único, que é usado para rotear pacotes de dados para seu destino.

- Roteadores:
  Roteadores são dispositivos responsáveis por encaminhar os pacotes de dados de um dispositivo para outro. Eles fazem parte da infraestrutura de rede e são usados para conectar redes locais (LANs) à Internet.

- Servidores:
  Servidores são computadores de alto desempenho que armazenam e disponibilizam informações na Internet.

Existem vários tipos de servidores, incluindo servidores web, servidores de e-mail, servidores de banco de dados, entre outros.

- DNS (Domain Name System):
  O DNS é um sistema que traduz nomes de domínio (por exemplo, www.example.com) em endereços IP. Isso permite que os usuários acessem sites e serviços usando nomes de fácil memorização em vez de números IP.

- World Wide Web (WWW):
  A World Wide Web é uma parte da Internet que consiste em páginas da web interconectadas. Ela é acessada através de navegadores da web e usa o protocolo HTTP (Hypertext Transfer Protocol) para transferir dados entre servidores e clientes.

- Protocolos de Comunicação:
  - A Internet utiliza uma variedade de protocolos de comunicação, como HTTP, FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol), POP3/IMAP (para e-mail), e muitos outros, para permitir a transferência de diferentes tipos de dados e informações.

- Pacotes de Dados:
  Os dados transmitidos pela Internet são divididos em pequenos pacotes. Cada pacote pode seguir caminhos diferentes pela rede e ser reagrupado no destino. Isso torna a comunicação mais eficiente e tolerante a falhas.

- Backbones da Internet:
  A Internet é composta por uma rede de infraestrutura que inclui cabos de fibra óptica submarinos, satélites e outras tecnologias de comunicação. Os provedores de Internet e as redes de backbone

(espinha dorsal) interconectam regiões geográficas e países.

- Segurança:
  A segurança é fundamental na Internet. Protocolos como HTTPS (HTTP seguro) e firewalls são usados para proteger a integridade e a privacidade dos dados transmitidos.

- Criptografia:
  A criptografia desempenha um papel crucial na Internet, especialmente em transações seguras e comunicações confidenciais. O uso de criptografia garante que os dados sejam protegidos contra interceptação e acessos não autorizados. O SSL/TLS é um protocolo comum usado para criptografar a comunicação em serviços online, como transações bancárias e compras online.

- Endereços IP:
  Existem dois tipos principais de endereços IP: IPv4 (Internet Protocol version 4) e IPv6 (Internet Protocol version 6). Com o esgotamento dos endereços IPv4, o IPv6 foi desenvolvido para fornecer um número significativamente maior de endereços IP, garantindo o crescimento contínuo da Internet.

- Peering e ISPs:
  A Internet é composta por uma rede de provedores de serviços de Internet (ISPs) que se conectam entre si por meio de acordos de peering. O peering permite o tráfego de dados entre diferentes partes da Internet, independentemente de qual ISP você usa.

- Redes de Entrega de Conteúdo (CDNs):
  As CDNs são sistemas distribuídos de servidores localizados em vários pontos geográficos. Elas armazenam cópias de conteúdo web, como imagens, vídeos e páginas da web, para acelerar a entrega e reduzir a latência.

Grandes empresas usam CDNs para melhorar o desempenho de seus sites e aplicativos.

O QUE É HTTP?

O HTTP (Hypertext Transfer Protocol) é um protocolo de comunicação usado para transferir dados na World Wide Web, a parte da Internet mais comumente acessada pelos usuários. Como um aspirante a desenvolvedor de backend, é fundamental entender o HTTP, pois é a base para a comunicação entre servidores e clientes da web. Aqui estão algumas informações essenciais sobre o HTTP:

- Protocolo Cliente-Servidor: O HTTP é um protocolo cliente-servidor, o que significa que ele facilita a comunicação entre um cliente (geralmente um navegador da web) e um servidor (onde um site ou aplicativo está hospedado).

- Métodos HTTP: O HTTP define vários métodos que indicam a ação que o cliente deseja realizar no servidor. Alguns dos métodos mais comuns incluem:
    - GET: Solicita dados de um recurso (por exemplo, uma página da web) do servidor.
    - POST: Envia dados para o servidor, geralmente usado para enviar informações de formulário.
    - PUT: Atualiza um recurso no servidor.
    - DELETE: Remove um recurso no servidor.

- Cabeçalhos HTTP: Os cabeçalhos HTTP são informações adicionais enviadas entre o cliente e o servidor para controlar a solicitação e a resposta. Eles podem incluir informações sobre o tipo de conteúdo, autenticação, cookies e muito mais.

- Códigos de Resposta HTTP: Quando o servidor responde a uma solicitação HTTP, ele retorna um código de resposta HTTP para indicar o resultado da solicitação. Alguns códigos comuns incluem:
    - 200 OK: A solicitação foi bem-sucedida.
    - 404 Not Found: O recurso solicitado não foi encontrado.
    - 500 Internal Server Error: O servidor encontrou um erro interno ao processar a solicitação.

- Stateless: O HTTP é um protocolo stateless, o que significa que cada solicitação do cliente é tratada independentemente e não mantém informações de estado entre solicitações. Para lidar com sessões e rastrear informações do cliente, técnicas como cookies ou tokens de autenticação são frequentemente usadas.

- Segurança: O HTTP não é seguro por padrão. No entanto, existe uma versão segura chamada HTTPS (Hypertext Transfer Protocol Secure), que usa criptografia para proteger a comunicação entre o cliente e o servidor. O uso do HTTPS é essencial para proteger os dados sensíveis dos usuários.

- REST e APIs: O HTTP é amplamente utilizado em serviços da web e APIs (Application Programming Interfaces). Um estilo comum de projeto de APIs web é baseado no REST (Representational State Transfer), que usa os métodos HTTP (GET, POST, PUT, DELETE) para realizar operações em recursos.

- Desenvolvimento de Backend: Como desenvolvedor de backend, você frequentemente trabalhará com servidores web que implementam o HTTP para fornecer serviços e recursos. Você escreverá código para processar solicitações HTTP, interagir com bancos de dados, autenticar usuários e fornecer respostas apropriadas.

- Versões do HTTP: Existem várias versões do protocolo HTTP, sendo as duas mais comuns o HTTP/1.1 e o HTTP/2. O HTTP/2 introduziu melhorias significativas no desempenho, como o uso de multiplexação e compactação de cabeçalhos, tornando as solicitações e respostas mais eficientes.
- APIs RESTful: Para desenvolvedores de backend, entender a arquitetura REST é essencial ao criar APIs web. O REST é um conjunto de princípios de design que promove a escalabilidade, simplicidade e flexibilidade das APIs. Isso inclui o uso adequado de URLs, métodos HTTP e manipulação de estados.

- Caching: O HTTP oferece suporte a caching, o que permite que os clientes armazenem em cache respostas do servidor para acelerar o carregamento de páginas e economizar largura de banda. Como desenvolvedor de backend, você pode configurar cabeçalhos de controle de cache para otimizar o desempenho.

- Autenticação e Autorização: Você precisará entender como implementar autenticação e autorização em aplicativos web e APIs. Isso envolve o uso de cabeçalhos de autorização, tokens de acesso e outras técnicas para garantir que apenas os usuários autorizados tenham acesso a recursos específicos.

- Segurança contra Ataques: Como desenvolvedor de backend, é fundamental entender e implementar práticas de segurança para proteger seu servidor contra ataques comuns, como injeção de SQL, XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery) e outros.

- Monitoramento e Desempenho: Monitore o desempenho de seus servidores e APIs para garantir que eles estejam funcionando eficientemente. Ferramentas de monitoramento, registro e análise de desempenho podem ser valiosas.

- Escalabilidade: À medida que seu aplicativo cresce, é importante entender como dimensionar seus servidores para lidar com um grande número de solicitações. Isso envolve a configuração de balanceadores de carga e a distribuição eficaz de recursos.
- Ferramentas e Frameworks: Existem muitas ferramentas e frameworks disponíveis para simplificar o desenvolvimento de aplicativos web e APIs em HTTP. Alguns exemplos incluem o Express.js para Node.js, Django e Flask para Python, Ruby on Rails e muitos outros.

BROWSERS E COMO FUNCIONAM

Os navegadores da web desempenham um papel crucial na experiência da Internet, permitindo que os usuários acessem e interajam com páginas da web, aplicativos web e recursos online. Eles funcionam como uma interface entre os usuários e os servidores da web. Aqui está uma visão geral de como os navegadores funcionam:

- Interface de Usuário: A interface do navegador é a parte que os usuários interagem diretamente. Ela inclui elementos como barras de endereço, botões de navegação (voltar, avançar, recarregar) e guias para a abertura de múltiplas páginas.

- Motor de Renderização: O motor de renderização é o componente central do navegador responsável por interpretar e renderizar o conteúdo HTML, CSS e JavaScript das páginas da web. Os navegadores populares, como o Chrome (Blink), Firefox (Gecko) e Safari (WebKit), têm seus próprios motores de renderização.

- Interpretação do HTML e CSS: O navegador começa baixando o HTML da página da web a partir do servidor. Ele interpreta o HTML para entender a estrutura da página, os links para recursos externos (como imagens e folhas de estilo CSS) e os elementos da página. Em seguida, ele solicita esses recursos do servidor.
- Carregamento de Recursos: Os navegadores carregam recursos adicionais, como imagens, folhas de estilo e scripts JavaScript, à medida que os encontram no HTML. Isso pode ser feito de forma paralela para acelerar o carregamento da página.

- Interpretação do JavaScript: Se a página contiver código JavaScript, o navegador o executará. Isso permite que os navegadores adicionem interatividade à página, respondendo a eventos do usuário e atualizando dinamicamente o conteúdo.

- Modelo de Objeto de Documento (DOM): O navegador cria um modelo de objeto de documento, conhecido como DOM, que representa a estrutura da página. O DOM é usado para manipular elementos da página usando JavaScript e para atualizar a interface do usuário em resposta a eventos.

- Modelo de Objeto de Navegação (BOM): Além do DOM, o navegador também cria um modelo de objeto de navegação (BOM) que fornece acesso ao histórico de navegação, janelas, guias e outras funcionalidades do navegador.

- Renderização: O motor de renderização converte o conteúdo interpretado (HTML, CSS e JavaScript) em uma representação visual na tela. Isso inclui o posicionamento de elementos, aplicação de estilos e renderização de imagens.

- Armazenamento em Cache: Os navegadores geralmente mantêm em cache recursos como imagens e arquivos CSS para acelerar o carregamento subsequente de páginas da web.

Isso permite que as páginas carreguem mais rapidamente quando você as visita novamente.

- Comunicação com Servidores: Os navegadores também lidam com a comunicação com servidores da web por meio do protocolo HTTP. Eles enviam solicitações para servidores para obter recursos e enviam dados do usuário de volta para o servidor, como ao preencher formulários.

- Segurança: Os navegadores implementam várias medidas de segurança, como a política de mesma origem (Same-Origin Policy), para proteger os usuários contra ameaças como ataques de script entre sites (XSS) e roubo de cookies.

- Extensões e Plugins: Os navegadores permitem que os usuários instalem extensões e plugins para personalizar e estender a funcionalidade do navegador. Isso inclui bloqueadores de anúncios, gerenciadores de senhas, tradutores e muito mais.

- Atualizações: Navegadores frequentemente lançam atualizações para melhorar o desempenho, a segurança e adicionar novos recursos. Manter o navegador atualizado é importante para manter a segurança e a compatibilidade com as páginas da web modernas.

- Compatibilidade com Padrões: Os navegadores devem ser compatíveis com padrões da web, como HTML, CSS e JavaScript, para garantir que as páginas da web sejam exibidas corretamente. No entanto, diferentes navegadores podem implementar os padrões de maneiras ligeiramente diferentes, o que pode levar a problemas de compatibilidade.

- Cookies: Os navegadores suportam cookies, que são pequenos pedaços de dados armazenados no lado do cliente. Os cookies são frequentemente usados para manter o estado

da sessão do usuário, rastrear informações de autenticação e personalizar a experiência do usuário.

- Armazenamento Local: Além de cookies, os navegadores oferecem mecanismos de armazenamento local, como o localStorage e o sessionStorage, que permitem aos desenvolvedores armazenar dados no dispositivo do usuário para uso posterior, mesmo quando o navegador está fechado.

- Web APIs: Os navegadores implementam várias APIs da web que permitem interações avançadas com o sistema e o dispositivo do usuário. Isso inclui acesso à câmera, microfone, geolocalização e notificações. Desenvolvedores podem usar essas APIs para criar aplicativos web ricos em recursos.

- Cross-Origin Resource Sharing (CORS): Para manter a segurança, os navegadores aplicam a política de mesma origem, que restringe a capacidade de uma página da web de fazer solicitações a um domínio diferente do seu próprio. O CORS é uma técnica que permite controlar as exceções a essa política e permitir solicitações cruzadas a servidores específicos.

- Motor JavaScript: Os navegadores contêm um mecanismo JavaScript (interpretador) que executa código JavaScript. Alguns dos motores JavaScript mais conhecidos incluem o V8 (utilizado no Chrome), o SpiderMonkey (utilizado no Firefox) e o JavaScriptCore (utilizado no Safari).

- Motor de Layout CSS: Além do motor de renderização, os navegadores possuem um motor de layout CSS que determina como os elementos HTML são estilizados e posicionados na

página. Isso inclui calcular tamanhos, cores, margens, preenchimentos e posições de elementos.

- Desenvolvedor e Ferramentas de Inspeção: Os navegadores oferecem ferramentas de desenvolvedor que permitem aos desenvolvedores inspecionar elementos, depurar JavaScript, monitorar solicitações de rede e otimizar o desempenho das páginas da web.

- Personalização e Extensões: Os navegadores modernos permitem que os usuários personalizem suas configurações, adicionem extensões e temas para adaptar a experiência de navegação às suas preferências.

- Modo de Navegação Anônima: A maioria dos navegadores oferece um modo de navegação anônima, que não mantém um histórico de navegação, cookies e outras informações após o encerramento da sessão.
- Páginas Web Progressivas (PWAs): Os navegadores suportam PWAs, que são aplicativos da web que oferecem uma experiência semelhante à de aplicativos nativos, incluindo notificações push, acesso offline e uma tela inicial personalizada.

DNS E COMO FUNCIONA

O Sistema de Nomes de Domínio (DNS, Domain Name System) é uma parte fundamental da infraestrutura da Internet que permite que os usuários acessem recursos online usando nomes de domínio amigáveis, em vez de ter que lembrar endereços IP numéricos. O DNS funciona como um diretório global que traduz nomes de domínio em endereços IP e vice-versa. Aqui está uma visão abrangente de como o DNS funciona:

- Hierarquia de Nomes de Domínio:
    - O DNS organiza nomes de domínio em uma hierarquia. No topo da hierarquia estão os domínios de nível superior (TLDs, Top-Level Domains), como .com, .org, .net, .gov e .edu. Abaixo dos TLDs, estão os domínios de segundo nível (SLDs), como exemplo.com. Essa estrutura hierárquica permite uma organização eficiente dos nomes de domínio.

- Zonas e Servidores de Nomes:
    - Cada domínio é gerenciado por um servidor de nomes (Name Server) responsável por armazenar informações sobre esse domínio. Os servidores de nomes podem ser autoritativos (contendo informações sobre um domínio específico) ou recursivos (encaminham solicitações de resolução de nomes para outros servidores).

- Consulta DNS:
    - Quando um usuário digita um nome de domínio na barra de endereços do navegador, o sistema operacional do computador do usuário faz uma consulta DNS para traduzir o nome de domínio em um endereço IP.
    - A consulta começa com o servidor DNS local do usuário, geralmente fornecido pelo ISP (provedor de serviços de Internet).
    - Se o servidor DNS local tem informações em cache para o nome de domínio solicitado, ele responde imediatamente. Caso contrário, ele encaminha a consulta para outros servidores DNS.

- Recursão e Autoridade:
    - O servidor DNS local consulta os servidores raiz para obter informações sobre os servidores de autoridade para o domínio de nível superior apropriado (.com, .org, etc.).
    - O servidor DNS local, em seguida, consulta os servidores de autoridade do TLD para obter

informações sobre o servidor de nomes autoritativo para o domínio de segundo nível.
    - Finalmente, o servidor DNS local consulta o servidor de nomes autoritativo do domínio de destino para obter o endereço IP associado ao nome de domínio.

- Cache DNS:
    - Para melhorar o desempenho e reduzir a carga nos servidores DNS, os servidores DNS mantêm em cache as informações de resolução de nomes por um período de tempo especificado, chamado de "tempo de vida" ou "TTL" (Time To Live). Isso significa que, se o mesmo domínio for acessado novamente em um curto período de tempo, o servidor DNS local pode usar as informações em cache em vez de realizar uma nova consulta.

- Atualizações de DNS:
    - A manutenção e a atualização do sistema DNS são de responsabilidade dos administradores de domínio. Eles podem atualizar as informações associadas aos seus domínios, como adicionar novos registros DNS (por exemplo, para e-mail ou servidores web) ou alterar as configurações de hospedagem.

- Segurança DNS:
    - O DNS é vulnerável a ataques, como ataques de envenenamento de cache (DNS cache poisoning) e ataques DDoS. Para proteger o sistema, várias técnicas e protocolos de segurança, como DNSSEC (DNS Security Extensions), são usados para verificar a autenticidade e a integridade dos registros DNS.

- Registro de Domínio:
    - Registrar um domínio envolve a compra do direito de uso de um nome de domínio específico por um determinado período de tempo. Organizações de registro de domínio, conhecidas como registradores,

facilitam esse processo e mantêm registros de nomes de domínio.

- Geolocalização e Balanceamento de Carga:
    - O DNS também é usado para realizar geolocalização e balanceamento de carga. Isso permite que os servidores DNS direcionem os usuários para servidores específicos com base em sua localização geográfica ou para distribuir o tráfego de forma uniforme entre vários servidores.

- DNS Dinâmico: Em algumas situações, como redes domésticas ou pequenos escritórios, os endereços IP atribuídos a computadores podem ser dinâmicos e mudar periodicamente. O DNS dinâmico permite que dispositivos atualizem automaticamente seus registros DNS quando o endereço IP muda.

- DNS Reverso: Além da tradução de nomes de domínio em endereços IP (resolução direta), o DNS também permite a resolução inversa, que traduz endereços IP em nomes de domínio. Isso é usado, por exemplo, para verificar a autenticidade de uma conexão de rede por meio de um registro PTR (Pointer Record).

- DNS Global: O DNS é uma infraestrutura global distribuída e replicada. Isso significa que ele é altamente redundante e resistente a falhas. Se um servidor DNS estiver inacessível, outros servidores podem fornecer as informações necessárias.

- Anycast DNS: Alguns serviços de DNS usam a técnica de Anycast para direcionar as solicitações dos clientes para

o servidor DNS mais próximo geograficamente. Isso melhora a latência e a eficiência da resolução de nomes.

- Registro SPF (Sender Policy Framework): O DNS também é usado para ajudar na prevenção de e-mails falsificados (spoofing). O registro SPF é um tipo de registro DNS que especifica quais servidores de e-mail têm permissão para enviar e-mails em nome de um domínio específico.
- Registro MX (Mail Exchange): Os registros MX são usados para direcionar e-mails para servidores de e-mail. Eles especificam os servidores de e-mail que devem receber mensagens destinadas a um domínio específico.

- Registro CNAME (Canonical Name): Um registro CNAME é usado para criar um alias para um nome de domínio. Isso permite que vários nomes de domínio se refiram ao mesmo local ou host.

- Registro SRV (Service): Os registros SRV são usados para identificar serviços disponíveis em um domínio, como servidores de mensagens instantâneas, servidores de voz sobre IP e outros serviços.

- Monitoramento de DNS: É importante monitorar a disponibilidade e o desempenho dos servidores DNS. Existem ferramentas e serviços que podem ajudar a garantir que o DNS funcione corretamente.

- DNS em Redes Privadas: Além do DNS público usado para acessar sites na Internet, as organizações frequentemente usam servidores DNS em redes privadas para gerenciar nomes e endereços IP internos.

- IPv6 e DNS: Com a adoção crescente do IPv6, o DNS desempenha um papel importante na resolução de nomes de domínio em endereços IPv6. Registros AAAA são usados para mapear nomes de domínio em endereços IPv6.

O QUE É DOMAIN NAME?

Um "Domain Name" (nome de domínio) é um identificador legível por humanos usado para localizar recursos na Internet, como sites, servidores de e-mail e outros serviços online. Como desenvolvedor de backend, você pode se deparar com nomes de domínio ao configurar servidores, trabalhar com serviços da web e desenvolver aplicativos. Aqui estão informações relevantes sobre domain names:

Estrutura de Nomes de Domínio:

- Os nomes de domínio são organizados em uma hierarquia que começa com um domínio de nível superior (TLD, Top-Level Domain), como .com, .org, .net, .gov, .edu e assim por diante. Abaixo dos TLDs, estão os domínios de segundo nível (SLD), como example.com. Essa estrutura é usada para organizar e identificar recursos na Internet.

Registro de Domínio:

- Para obter um nome de domínio específico, como example.com, um indivíduo ou organização precisa registrá-lo por meio de registradores de domínio autorizados. O registro de domínio envolve o pagamento de uma taxa e a especificação de informações de contato.

Whois:

- O Whois é um serviço que permite consultar informações sobre a propriedade de um domínio, incluindo o registrante, informações de contato e datas de registro e vencimento. Isso é útil para verificar a propriedade de um domínio e entrar em contato com o registrante.

Registro de Domínio Internacionalizado (IDN):

- IDNs permitem o uso de caracteres não latinos em nomes de domínio. Isso torna possível registrar domínios em idiomas não baseados em caracteres latinos, como árabe, chinês ou russo.

DNS e Nomes de Domínio:

- O Sistema de Nomes de Domínio (DNS) é responsável por traduzir nomes de domínio em endereços IP, permitindo que os navegadores e outros aplicativos encontrem recursos na Internet. Isso é essencial para a navegação na web e para a comunicação online.

Subdomínios:

- Além dos domínios de segundo nível, os subdomínios permitem que uma organização crie estruturas adicionais na hierarquia do domínio. Por exemplo, sub.example.com é um subdomínio de example.com.

Registro e Hospedagem Separados:

- O registro de domínio e a hospedagem de sites são serviços separados. Você pode registrar um domínio com um provedor e hospedar o site em um servidor diferente. Isso oferece flexibilidade para gerenciar recursos da web.

Redirecionamentos de Domínio:

- Os redirecionamentos de domínio permitem que um domínio encaminhe o tráfego para outro domínio. Isso é útil ao migrar um site, alterar um nome de domínio ou criar redirecionamentos personalizados.

Segurança de Domínio:

- É importante proteger seus domínios contra roubo ou uso não autorizado. Isso envolve a configuração de

autenticação de dois fatores (2FA), uso de senhas fortes e monitoramento regular.

Transferência de Domínio:

- Os domínios podem ser transferidos de um registrador para outro. Isso é comum ao mudar de provedor de hospedagem ou ao vender um domínio a outra pessoa.

Registros de DNS:

- Além do registro A (para endereços IPv4) e AAAA (para endereços IPv6), existem outros tipos de registros DNS, como registros MX (para servidores de e-mail), registros CNAME (para aliases), registros TXT (para informações de texto) e registros SRV (para serviços).

Renovação de Domínio:

- Os domínios geralmente são registrados por um período de tempo específico (por exemplo, um ano). Para continuar usando um domínio, ele deve ser renovado antes do vencimento.

Política de Privacidade de Domínio:

- Alguns registradores oferecem serviços de privacidade de domínio, que ocultam as informações de contato do registrante nos registros Whois públicos, fornecendo anonimato adicional.

Registros SPF (Sender Policy Framework):

- Registros SPF são usados para ajudar a prevenir e-mails falsificados (spoofing). Eles especificam quais servidores de e-mail estão autorizados a enviar e-mails em nome de um domínio específico.

Registros DKIM (DomainKeys Identified Mail):

- Registros DKIM são usados para assinar digitalmente e-mails enviados de um domínio, permitindo que os

destinatários verifiquem a autenticidade das mensagens. Isso ajuda a reduzir o risco de spoofing e phishing.

Registro DNSSEC (DNS Security Extensions):

- O DNSSEC é um conjunto de extensões de segurança que ajuda a proteger a integridade dos registros DNS. Ele permite a verificação da autenticidade das informações de DNS, ajudando a prevenir ataques de envenenamento de cache DNS.

Renovação Automática de Domínio:

- Muitos registradores oferecem serviços de renovação automática de domínio, para evitar que um domínio expire inadvertidamente. Isso é útil para garantir que você não perca a propriedade de um domínio importante.

Registro de Domínio com Caracteres Especiais:

- Em alguns casos, é possível registrar domínios com caracteres especiais, como acentos, caracteres não latinos ou emojis. Esses domínios são chamados de IDNs (Internationalized Domain Names).

Transferência Segura de Domínio:

- Ao transferir um domínio para outro registrador, é importante seguir um processo seguro e verificar a autenticidade para evitar possíveis transferências não autorizadas.

Registro de Domínio de Nível Superior Personalizado (ccTLDs):

- Além dos TLDs genéricos (gTLDs) como .com, .org e .net, existem domínios de nível superior personalizados (ccTLDs) associados a países ou territórios específicos. Cada ccTLD tem suas próprias regras e requisitos de registro.

Uso de Subdomínios para Aplicativos e Serviços: Subdomínios são frequentemente usados para direcionar tráfego para aplicativos e serviços específicos. Por exemplo, app.example.com pode ser usado para um aplicativo web separado de www.example.com.

Nomes de Domínio para Redes Internas (DNS Privado):

- Em redes internas, nomes de domínio personalizados podem ser usados para identificar recursos de rede, como servidores e dispositivos. Isso é conhecido como DNS privado.

Integração de Domínio com Serviços na Nuvem:

- Ao implantar aplicativos e serviços em nuvens públicas, como AWS, Azure ou Google Cloud, é comum integrar nomes de domínio personalizados para acessar recursos na nuvem.

- Políticas de Registro de Domínio do Registrador: Cada registrador de domínio pode ter suas próprias políticas e termos de serviço. É importante entender as políticas do registrador escolhido ao registrar um domínio.

- Monitoramento de Domínio: Monitorar a disponibilidade de domínios, datas de expiração e registros DNS é fundamental para garantir que os domínios permaneçam operacionais e em conformidade.

O QUE É HOSTING?

"Hosting" (hospedagem) refere-se ao serviço de armazenar e disponibilizar recursos, como sites, aplicativos, arquivos e dados, em servidores ou infraestrutura de computação remota para que eles possam ser acessados pela Internet. É um componente fundamental para a presença online de empresas e indivíduos. Aqui está uma visão abrangente de hospedagem:

1. **Tipos de Hospedagem**:
   - Existem vários tipos de hospedagem, cada um atendendo a diferentes necessidades e orçamentos. Os principais tipos incluem:
     - **Hospedagem Compartilhada**: Vários sites compartilham os recursos de um servidor, o que a torna uma opção econômica, mas com recursos limitados.
     - **Hospedagem VPS (Virtual Private Server)**: Um servidor físico é dividido em máquinas virtuais, oferecendo mais controle e recursos do que a hospedagem compartilhada.
     - **Hospedagem Dedicada**: Um servidor inteiro é dedicado a um único cliente, proporcionando o máximo de controle e recursos.
     - **Hospedagem em Nuvem**: Os recursos de hospedagem são provisionados em um ambiente de nuvem, permitindo escalabilidade e flexibilidade.
     - **Hospedagem WordPress**: Especializada em hospedar sites construídos em WordPress, oferecendo recursos otimizados para essa plataforma.

2. **Provedores de Hospedagem**:
   - Existem muitos provedores de hospedagem no mercado, como Bluehost, HostGator, SiteGround, AWS, Google Cloud, Microsoft Azure e muitos outros. A escolha do provedor depende das necessidades específicas e do orçamento do projeto.

3. **Servidores e Data Centers**:
   - Os provedores de hospedagem possuem e operam servidores e data centers, que são instalações de infraestrutura de computação onde os servidores são mantidos. Isso envolve garantir energia confiável, resfriamento adequado e segurança física.

4. **Armazenamento e Backup**:
   - Os recursos hospedados incluem dados, arquivos e bancos de dados. Provedores de hospedagem geralmente oferecem soluções de armazenamento e backup para proteger esses ativos.

5. **Banda Larga e Conectividade**:
   - Provedores de hospedagem garantem conectividade confiável à Internet para que os recursos hospedados estejam sempre acessíveis aos usuários.

6. **Sistemas Operacionais e Servidores Web**:
   - Os servidores usam sistemas operacionais (como Linux ou Windows) e servidores web (como Apache, Nginx ou IIS) para servir conteúdo e aplicativos pela Internet.

7. **Hospedagem Gerenciada e Não Gerenciada**:
   - Alguns provedores de hospedagem oferecem serviços gerenciados em que eles cuidam da manutenção, atualizações e segurança do servidor. A hospedagem não gerenciada requer que os clientes administrem seu próprio servidor.

8. **Segurança**:
   - A segurança é uma preocupação importante na hospedagem. Isso inclui medidas para proteger os dados armazenados e transmitidos, como firewalls, criptografia SSL e detecção de intrusões.

9. **Escalabilidade**:
   - A hospedagem em nuvem oferece escalabilidade, permitindo que os recursos sejam dimensionados automaticamente de acordo

com a demanda. Isso é particularmente útil para sites e aplicativos com tráfego variável.

10. **Uptime e SLA (Service Level Agreement)**:
    - A disponibilidade é fundamental. Os provedores de hospedagem geralmente garantem um determinado nível de tempo de atividade em seus SLAs, o que é importante para manter a confiabilidade do serviço.

11. **Suporte Técnico**:
    - Um bom suporte técnico é crucial. Os provedores de hospedagem oferecem diferentes níveis de suporte, desde suporte 24/7 por telefone até recursos de autoatendimento.

12. **Migração de Hospedagem**:
    - Mudar de provedor de hospedagem é possível, mas requer cuidados na migração de dados e configurações para evitar interrupções no serviço.

13. **Hospedagem de E-mail**:
    - Além de hospedar sites, muitos provedores oferecem serviços de hospedagem de e-mail, permitindo que os clientes criem contas de e-mail personalizadas com seus domínios.

14. **Preços e Contratos**:
    - Os preços de hospedagem variam dependendo do tipo de hospedagem, recursos incluídos e provedor. Contratos podem ser mensais, anuais ou de longo prazo.

15. **Recursos Otimizados**:
    - Alguns provedores oferecem recursos otimizados para plataformas específicas, como WordPress, Joomla ou lojas online, para melhor desempenho e segurança.

16. **CDNs (Content Delivery Networks)**:
    - CDNs são usadas em conjunto com a hospedagem para acelerar a entrega de conteúdo, como imagens e vídeos, a usuários em todo o mundo.

17. **Registros DNS e Gerenciamento de Domínio**:
    - Muitos provedores de hospedagem oferecem recursos de gerenciamento de registros DNS, permitindo a configuração de redirecionamentos, subdomínios e outros aspectos relacionados ao domínio.

A hospedagem é um aspecto crítico do funcionamento de sites e aplicativos na Internet. Como desenvolvedor de backend, é importante entender os conceitos básicos de hospedagem, pois você provavelmente precisará interagir com servidores, configurar recursos de hospedagem e garantir que os serviços estejam disponíveis para os usuários finais. Escolher a hospedagem certa é fundamental para garantir a escalabilidade, desempenho e segurança de suas aplicações e recursos online.

OS E CONHECIMENTO GERAL

POSIX BASIC (stdin, stdout,stderr,pipes)

O padrão POSIX (Portable Operating System Interface) é um conjunto de padrões definidos para sistemas operacionais Unix-like e foi projetado para garantir a compatibilidade entre esses sistemas. Os conceitos de stdin, stdout, stderr e pipes são fundamentais no contexto da programação POSIX e sistemas Unix-like. Vou explicar cada um deles:

stdin (Standard Input):

- stdin é um fluxo de entrada padrão que representa a entrada de dados em um programa ou processo. Por padrão, stdin está conectado ao teclado, o que significa que você pode inserir dados por meio do teclado enquanto o programa está sendo executado. No entanto, você também pode redirecionar stdin para

ler dados de arquivos ou outros dispositivos de entrada.

Exemplo em C:

```
#include <stdio.h>

int main() {
 char buffer[100];
 printf("Digite algo: ");
  fgets(buffer, sizeof(buffer), stdin); // lê uma linha do teclado
 printf("Você digitou: %s", buffer);
 return 0;
}
```

stdout (Standard Output):

- stdout é um fluxo de saída padrão que representa a saída de dados de um programa ou processo. Por padrão, stdout está conectado à tela (console), o que significa que os dados impressos no stdout são exibidos no terminal. Assim como stdin, stdout também pode ser redirecionado para escrever dados em arquivos ou outros dispositivos de saída.

Exemplo em C:

```
#include <stdio.h>

int main() {
 printf("Isso é uma saída padrão.\n");
 return 0;
}
```

stderr (Standard Error):

- stderr é semelhante a stdout, mas é usado para a saída de erros. Quando um programa encontra um erro, ele pode escrever mensagens de erro em stderr. Como stderr também está conectado à tela por padrão, as

mensagens de erro geralmente são exibidas no terminal. No entanto, assim como stdin e stdout, você pode redirecionar stderr para arquivos ou outros destinos.

Exemplo em C:

```
#include <stdio.h>

int main() {
 fprintf(stderr, "Isso é uma mensagem de erro.\n");
 return 1; // Retorna um código de saída não zero para indicar erro.
}
```

Pipes:

- Os pipes são mecanismos para encadear a saída de um processo à entrada de outro processo. Eles permitem a comunicação entre processos por meio da transferência de dados de stdout de um processo para stdin de outro processo. Os pipes são frequentemente usados para criar fluxos de dados entre programas, possibilitando a construção de pipelines de processamento de dados.

Exemplo no shell Unix (Linux, macOS):

sh

```
cat arquivo.txt | grep palavra
```

Nesse exemplo, o comando cat lê o conteúdo do arquivo.txt e envia a saída para grep, que procura a palavra "palavra". O pipe (|) conecta a saída de cat à entrada de grep.

Esses conceitos são fundamentais na programação POSIX e em sistemas Unix-like, pois permitem a entrada, saída e processamento de dados de maneira flexível e eficiente. Eles

são amplamente usados na criação de scripts, programação de sistemas, automação e muitos outros casos de uso. Compreender como trabalhar com stdin, stdout, stderr e pipes é essencial para programadores que trabalham em ambientes Unix-like.

COMANDOS BÁSICOS DO TERMINAL

Comandos Básicos:

ls: Listar arquivos e diretórios no diretório atual.
pwd: Mostrar o diretório de trabalho atual.
cd: Mudar de diretório.
touch: Criar um novo arquivo vazio.
mkdir: Criar um novo diretório.
rm: Remover arquivos ou diretórios.
cp: Copiar arquivos e diretórios.
mv: Mover ou renomear arquivos e diretórios.
cat: Exibir ou concatenar o conteúdo de arquivos.
less ou more: Visualizar conteúdo de arquivos página por página.
head e tail: Exibir o início ou o final de um arquivo.
find: Pesquisar arquivos e diretórios.
grep: Pesquisar texto em arquivos.
chmod: Alterar permissões de arquivos e diretórios.
chown: Alterar o proprietário de arquivos e diretórios.
ps: Listar processos em execução.
kill: Encerrar processos.
top ou htop: Monitorar recursos do sistema e processos em tempo real.
df: Exibir informações sobre o uso de disco.
du: Exibir o tamanho de diretórios e arquivos.

free: Exibir informações sobre a memória do sistema.

Comandos de Redes:

ifconfig ou ip: Configurar e exibir informações sobre interfaces de rede.
ping: Testar a conectividade de rede.
netstat ou ss: Exibir informações sobre portas e conexões de rede.
ssh: Conectar-se a servidores remotos de forma segura.
scp: Copiar arquivos de ou para um servidor remoto com segurança.

Comandos de Arquivos Compactados:

tar: Arquivar e desarquivar arquivos.
gzip: Comprimir e descomprimir arquivos.
zip e unzip: Compactar e descompactar arquivos no formato ZIP.

Comandos de Texto:

cat (já mencionado): Exibir ou concatenar o conteúdo de arquivos.
sort: Classificar linhas de um arquivo de texto.
uniq: Remover linhas duplicadas de um arquivo de texto.
cut: Recortar seções de um arquivo de texto.
sed: Editor de fluxo para manipulação de texto.
awk: Processador de texto e dados.

Comandos de Usuários e Grupos:

passwd: Alterar a senha de um usuário.
useradd: Adicionar um novo usuário.
userdel: Remover um usuário.

groupadd: Adicionar um novo grupo.
usermod e groupmod: Modificar propriedades de usuários e grupos.

Comandos de Serviços e Processos:

systemctl: Gerenciar serviços do sistema (systemd).
service: Controlar serviços do sistema init.
ps (já mencionado): Listar processos em execução.

Comandos de Agendamento de Tarefas:

crontab: Gerenciar tarefas agendadas (cron jobs).

Comandos de Pacotes:

apt (Debian/Ubuntu) ou yum (Red Hat/CentOS): Gerenciar pacotes do sistema.
dpkg (Debian/Ubuntu) ou rpm (Red Hat/CentOS): Gerenciar pacotes individualmente.

Comandos de Logs:

tail (já mencionado): Exibir o final de arquivos de log.
journalctl: Exibir logs do sistema (systemd).
dmesg: Exibir mensagens do kernel.

Comandos de Disco e Partições:

fdisk e parted: Gerenciar partições de disco.
mount e umount: Montar e desmontar sistemas de arquivos.

Comandos de Rede e DNS:

ping (já mencionado): Testar a conectividade de rede.
nslookup ou dig: Consultar informações de DNS.
ifconfig ou ip (já mencionado): Configurar interfaces de rede.

Comandos de Acesso Remoto:

ssh (já mencionado): Conectar-se a servidores remotos.
rsync: Sincronizar arquivos entre sistemas remotos.

Comandos de Compressão e Descompressão:

tar (já mencionado): Arquivar e desarquivar arquivos.
gzip (já mencionado): Comprimir e descomprimir arquivos.
bzip2: Comprimir e descomprimir arquivos no formato bzip2.
xz: Comprimir e descomprimir arquivos no formato xz.

Esses são alguns dos comandos mais usados em sistemas Linux e Unix-like. Eles são essenciais para a administração do sistema, gerenciamento de arquivos, redes, segurança e automação de tarefas. Conhecer esses comandos é fundamental para qualquer administrador de sistema ou usuário que trabalhe com sistemas Linux.

SISTEMAS OPERATIVOS

O que é um sistema operacional?

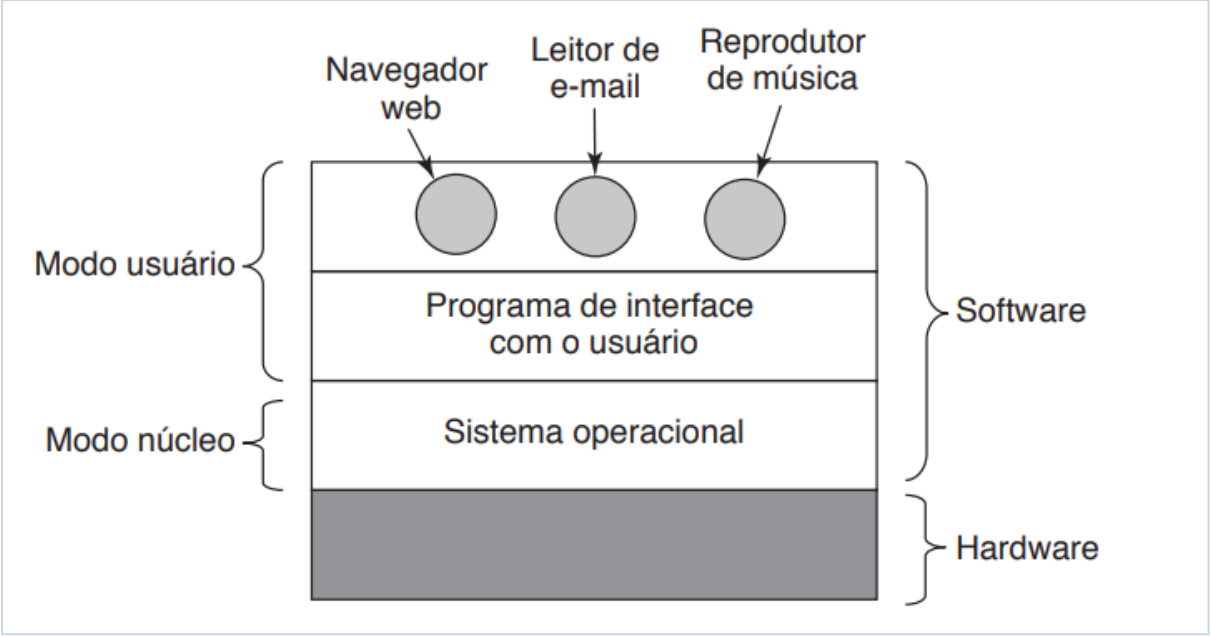
Fundamentalmente, um sistema operacional é um software, que pode ser o Linux, Windows, Android, macOS, UNIX, entre outros. No entanto, ele não resume aquilo que seus olhos conseguem ver ou ao que você consegue interagir. Em outras palavras, é um programa que conversa diretamente com o hardware da sua máquina.
O sistema operacional assegura que os programas funcionem corretamente. Mas antes de entendermos o que é um sistema operacional, precisamos definir o que é um sistema computacional.
Um computador moderno apresenta alguns elementos principais, que podemos destacar:

- Um ou mais processadores (o core);
- Memória principal;
- Dispositivos de entrada e saída (E/S), como monitores e teclados.

Todos esses componentes lidam em conjunto com os programas na sua máquina. À primeira vista pode parecer pouca coisa, mas você conhece de fato todos os componentes que pertencem ao seu computador? Sabe como todos eles funcionam?
Já pensou então, se todo o usuário e usuária se preocupassem com detalhes de hardware para escrever um arquivo de texto? Ou pior, se os desenvolvedores e desenvolvedoras tivessem que aprender os pormenores dos componentes de um computador moderno para conseguir começar a programar? Parece um trabalho sem fim, não é?

Qual a função do sistema operacional?

O autor do livro sistemas operacionais modernos, Tanenbaum, levanta essa questão e justifica que o sistema operacional surgiu para intermediar o diálogo entre máquina e pessoa programadora (ou mesmo usuário/a mais avançado).
Assim, a gente não precisa se preocupar em administrar todos os recursos na mão, ou então, por exemplo, ter que deslocar um espaço na memória específico para determinada variável?
Ademais, certamente a quantidade de programadores e softwares disponíveis na atualidade não seria a mesma se houvesse a necessidade também de gerenciar formas de otimização dos recursos.
Muitas vezes o sistema operacional é considerado um software base, porque fornece suporte às aplicações. Ele funciona como uma interface entre a aplicação e as rotinas de E/S com o hardware, ou seja, é preciso ter um sistema que faça essa conversa, e é nesse campo que o sistema operacional atua. Essa peça mais elementar de software opera em dois modos: o modo núcleo (modo supervisor) e o modo usuário.
No modo supervisor o acesso é completo a todo o hardware, e é possível enviar qualquer instrução que a máquina seja capaz de executar. O modo usuário é mais restrito para instruções que interferem no controle da máquina, mas é onde todo o resto do software opera.
Na imagem abaixo, conseguimos visualizar de forma simplificada as responsabilidades dos modos usuário e núcleo, assim como o local de atuação dos programas de interface com o usuário (GUI ou shell):



Fonte: Sistemas Operacionais Modernos. Tanenbaum, p.1.

Embora a divisão de responsabilidades pareça muito clara, em diversas ocasiões, a distinção entre o campo de atuação do sistema operacional e o modo usuário (software usual) se torna complexa. Um exemplo dessa afirmação é um programa que funciona em modo usuário, mas executa tarefas privilegiadas, como permitir ao usuário a troca de senha.
Em outras palavras, definir exatamente um sistema operacional não é tão simples assim, mas conseguimos diferenciá-los de programas de usuário por meio de suas particularidades.
Em resumo, as principais funções do sistema operacional são:

- 1) funcionar como uma ponte entre aplicações na camada do usuário e hardware;
- 2) e gerenciar os recursos de um sistema complexo (por exemplo, quando você executa vários programas ao mesmo tempo, na realidade é o seu sistema operacional que troca, em frações de segundos, o programa processado pela unidade central de processamento).

Além disso, são programas robustos, complexos e possuem longa vida. Apesar de serem muito difíceis de escrever, você acredita que o Linux tem cerca de cinco milhões de linhas? E isso é só a parte que opera no modo núcleo!

Sistema operacional, o herói mascarado

Já discutimos que lidar com a arquitetura de hardware é algo muito trabalhoso, a linguagem de máquina é complexa e, por essa razão, todos os sistemas operacionais fornecem ao usuário um nível de abstração.

Para criar um arquivo, uma foto ou música no seu computador, você não precisa conhecer os pormenores de hardware, mas usa a abstração fornecida para realizar essa tarefa. Dessa maneira, o hardware fica “escondido” nesse processo.

Os sistemas operacionais também fornecem uma série de benefícios, como maior portabilidade das aplicações (possibilidade de uso de uma mesma aplicação em diferentes computadores), e proporcionam o espaço para que as aplicações se dediquem a problemas de alto nível, ou seja, aos problemas que foram designadas para solucionar.

Além disso, só é possível executar múltiplos programas simultaneamente graças ao gerenciamento de recursos do sistema operacional.
O gerenciamento de recursos apresenta a multiplexação, que é o compartilhamento de recursos no tempo e no espaço. Em outras palavras, quando diferentes programas ou mesmo usuários realizam o revezamento no uso de um recurso, esse recurso é multiplexado no tempo.

Um exemplo bem comum de multiplexação no tempo é quando temos várias saídas de impressão em fila e apenas uma impressora, o que ocorre é que haverá uma decisão sobre qual documento será impresso primeiro.

Evolução dos Sistemas operacionais

Com o contínuo avanço da tecnologia, os sistemas operacionais assumiram um papel ainda mais fundamental na nossa vida cotidiana.

Os sistemas operacionais hoje, na 5ª geração de computadores, não se limitam aos desktops, mainframes ou notebooks. Observamos a presença em diferentes dispositivos, como IoT, celulares, até carros autônomos e muitos outros.

Diversas arquiteturas de sistemas operacionais foram criadas para atender a essas demandas especializadas. Vamos conhecer alguns?

Tipos e estruturas de Sistemas operacionais

Assim como existem dispositivos diferentes, também há tipos de sistemas operacionais específicos, com estruturas que se encaixam melhor em determinados computadores. Essas distintas formas de processamento do sistema operacional, estão divididas em monoprogramada e multiprogramada.
Formas de processamento:

- Monoprogramada ou serial: um único programa na máquina.
    - Exemplo: embarcados.
- Multiprogramada ou concorrente: é eficiente e apresenta vários programas dentro de um sistema.
    - Exemplo: Linux, Windows, UNIX.

É através das chamadas de sistemas (System Calls) que executamos ou interrompemos os processos. Um exemplo é a chamada mkdir no terminal, que cria um novo diretório.

Agora que compreendemos de forma sucinta como ocorre a atuação dos sistemas operacionais, é interessante observarmos também algumas estruturas que já foram implementadas na prática, são elas: sistema monolítico, sistemas de camadas, micronúcleos, modelo cliente-servidor, máquinas virtuais e exonúcleos.

Dentre os sistemas citados, podemos destacar alguns que usualmente aparecem em nossa rotina:

- Sistema monolítico: possui um modo núcleo e um modo usuário. “O sistema é escrito como uma coleção de

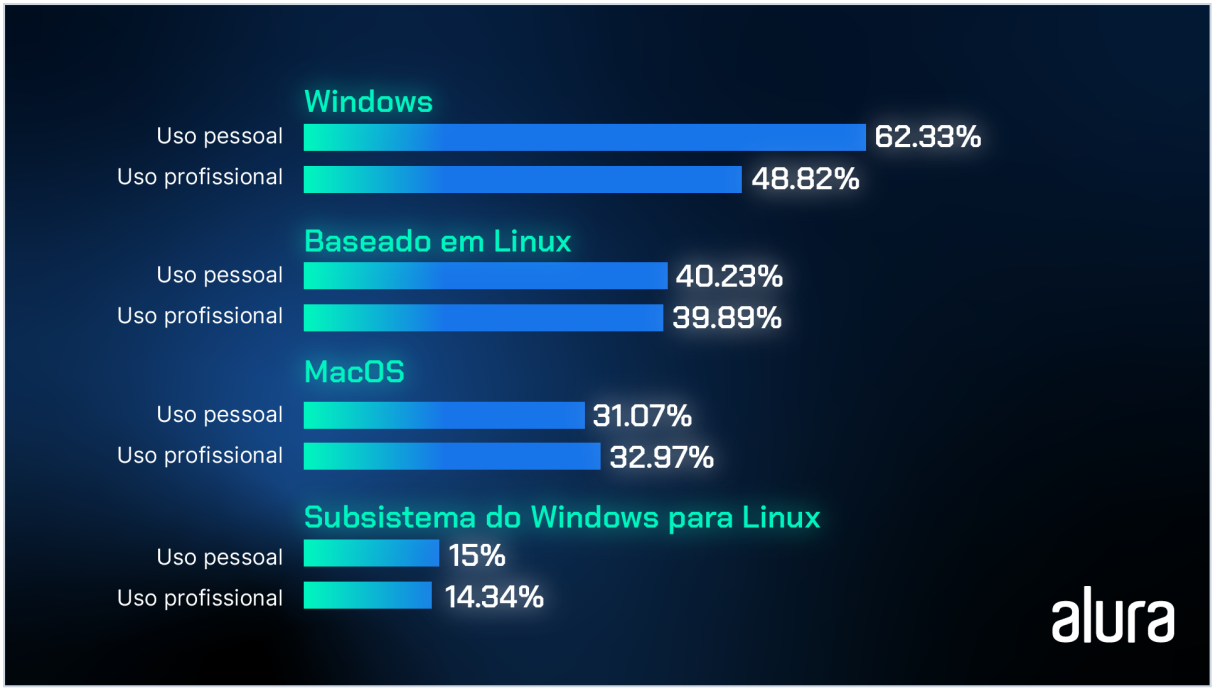
rotinas, ligadas a um único grande programa binário executável" (Tanenbaum, p.44).
    - Exemplos de sistemas operacionais: Linux, UNIX, Windows.
- Sistemas de camadas: é modular, isso significa que sua forma de operar é através da divisão de funcionalidades que correspondem a uma hierarquia.
    - Exemplos: MULTICS, OpenVMS.
- Micronúcleo ou microkernel: o objetivo da arquitetura de kernel é atingir alta confiabilidade por meio da divisão do sistema em pequenos módulos, onde apenas um é executado em modo núcleo, o micronúcleo, e o restante funciona em processos comuns.
    - Exemplos: Symbian, MINIX 3
- Virtual Machine, ou Máquina Virtual: é basicamente a virtualização de um outro sistema dentro do seu sistema operacional, possui uma camada limiar que faz.
    - Exemplo: Máquina Virtual Java.
    -

Os sistemas operacionais são um universo!

Quais os sistemas operacionais mais usados?
Em 2022, a Stack Overflow realizou uma pesquisa abrangente, na qual uma das perguntas feitas era: "Qual é o sistema operacional principal em que você trabalha?"

Na pesquisa realizada, o Windows se destacou como o sistema operacional mais amplamente adotado entre os desenvolvedores, tanto para fins pessoais quanto profissionais. Além disso, constatou-se que sistemas operacionais com base em Linux registraram uma popularidade superior à do macOS.

Para o Windows, 62,33% responderam que optam por uso pessoal, e 48,82% para uso profissional. Já o Linux, 40,23% para uso pessoal e 39,89% profissional.
Confira no gráfico abaixo:



Créditos:Stack Overflow

O Windows é normalmente escolhido para uso pessoal devido à familiaridade prévia das pessoas usuárias , sendo o sistema operacional padrão na maioria dos dispositivos.

Conclusão

Se conhecer circuitos, dispositivos de entrada e saída, monitor, memória, processador, e demais elementos em detalhes fosse requisito para escrever as linhas de código, sem dúvida nenhuma quase nenhum programa seria feito, como esclarece Tanenbaum.

Os sistemas operacionais então, surgem como programas fundamentais para operar diferentes computadores e garantir maior eficiência e experiência para o usuário e usuária.
Neste apanhado geral, conseguimos vislumbrar o que é sistema operacional e o quanto ele é fascinante. Além disso, também foi possível compreender para que serve o sistema operacional e quais os tipos existentes.

Sabemos agora, que a escolha do sistema operacional também está relacionada à sua necessidade e função do seu computador, ou seja: 1) instale diferentes sistemas operacionais (uma máquina virtual já é um excelente começo!); 2) avalie suas funcionalidades e, por fim; 3) estude seus pormenores, pois os sistemas operacionais são realmente impressionantes. E lembre-se, a inteligência e utilidade no uso da tecnologia quem realiza é você!

GERENCIAMENTO DE MEMÓRIA

Gerenciamento (ou gestão) de memória é um complexo campo da ciência da computação e são constantemente desenvolvidas várias técnicas para torná-la mais eficiente. Em sua forma mais simples, está relacionado em duas tarefas essenciais:

- Alocação: Quando o programa requisita um bloco de memória, o gerenciador o disponibiliza para a alocação;
- Reciclagem: Quando um bloco de memória foi alocado, mas os dados não foram requisitados por um determinado número de ciclos ou não há nenhum tipo de referência a este bloco pelo programa, esse bloco é liberado e pode ser reutilizado para outra requisição.

Gerência de Memória

A cada dia que passa os programadores necessitam de mais memória e mais programas rodando simultaneamente para poderem tratar cada vez mais informações. O tratamento necessário da memória utilizada não é uma tarefa fácil de ser implementada. Existem vários requisitos que devem ser observados para o correto funcionamento, tais como, segurança, isolamento, performance, entre outros. Para isto a função de gerenciar a memória passa a ser do sistema operacional e não mais do aplicativo.

Para que uma memória funcione de maneira correta, é necessário que se tome cuidado com vários elementos como segurança e isolamento, e para isso é utilizado o gerenciamento de memória. Este desenvolve sua função a partir de duas tarefas, a Alocação de Memória e a Fragmentação:

- A Alocação pode ser tanto estática, feita quando o programa é compilado, e a dinâmica, adiada até a execução.
- A Fragmentação, desperdício de memória, por sua vez pode ser interna, sobra na memória reservada ao programa, e externa que acontece quando após o termino dos programas são deixadas pequenas lacunas entre as páginas.

Para que a utilização da memória seja mais vantajosa, é utilizada a Paginação, processos virtuais da memória, aplicados na divisão da memória física em partições menores, chamadas de frames. O conjunto de registradores especiais rápidos chama-se Translation Lookaside Buffer, estes são subdivididos em chave valor que lhe é dado em todos os registradores ao mesmo tempo, e valor.
Existe uma técnica de gerencia de memória chamada memória virtual, que é onde memórias principais e secundárias juntas criam a ilusão de que há muito mais memória.
O conceito desta técnica fundamenta-se em não vincular o endereçamento feito pelo programa aos endereços físicos da memória principal, com isso os programas e suas estruturas de dados não se limitam ao tamanho da memória física, e assumem endereços na memória secundária.

O gerenciamento de memória virtual pode ocasionar vazamento de memória, ou seja, quando determinada quantia de memória é alocada e não é liberada mesmo que não sendo utilizada, assim dados perdem a referencia sem ao menos terem usado memória.
O gerenciamento automático chama-se Garbage collector. Ele retira os blocos de memória automaticamente. Seus algoritmos são divididos em duas famílias: a Identificação direta, por contagem de referência, e a Identificação indireta, por varrimento.

Alocação de Memória

Todo programa precisa utilizar memória para ser executado. Quando um programa inicia sua execução, ele começa a solicitar memória ao sistema operacional, ou seja, faz a alocação de memória necessária para a sua execução. A alocação de memória está dividida em 3(três) partes:

1. **Alocação Estática:** Decisão tomada quando o programa é compilado. Quando o programa é executado o Sistema operacional o lê e cria um processo, sendo o programa uma noção estática e o processo o programa em execução, que é criado em armazenamento primário e após isso recebe um espaço na memória. O espaço de memória é dividido em varias partes, uma das partes se chama segmentos de memória, que armazena dados estáticos, e outro se chama segmento de código que guarda instruções do programa. Quando o programa é executado o registrador PC apontará para determinado endereço do segmento de código do processo, que se chama TEXT. Para que se realize a alocação estática o compilador deve saber o total de memória que está livre, mandar esta informação para o S.O para que este crie um segmento de dados.

2. **Alocação Dinâmica:** Decisão é adiada até a execução. (Permite Swapping) Os objetos alocados dinamicamente podem ser criados e liberados a qualquer momento, em qualquer ordem, o que difere dos objetos locais das funções, que são criados e destruídos em uma ordem específica. Dado isto, é preciso organizar a memória para objetos dinâmicos de uma forma que possibilite a gestão do tempo de vida dos objetos por parte do programador. A memória reservada para objetos dinâmica costuma ser chamada de heap, existem várias formas de organizar um heap. Em linguagens sem a gestão automático (linguagem C), da memória dinâmica, uma organização usual do heap é uma lista encadeada de blocos livres, porém este tipo de

organização pode ter problemas devido à fragmentação dos blocos. Já em linguagens com gerenciamento automático de memória dinâmica (Java), a organização do heap depende da parte do sistema de tempo de execução encarregada desta gestão. Este componente é normalmente chamado de coletor de lixo. Alguns tópicos para podermos entender melhor: • Carrega vários processos na memória ao mesmo tempo; • Quando chega um novo processo, e a memória principal está toda ocupada, escolhe um processo, grava-o no disco, e libera memória para o próximo.

3. **Alocação Local:** Este processo de alocação é usado para variáveis que são locais a funções e sub-rotinas. Isso significa que o processo em execução deve manter acessível as variáveis locais da função ou procedimento que está executando no momento. Além disso, pelas propriedades do escopo em blocos, também devem estar acessíveis as variáveis de blocos mais externos. Em linguagens que permitem a definição de funções aninhadas, ter acesso a variáveis de quaisquer funções definidas externamente à função atualmente em execução. Como uma função pode chamar outras funções, um número arbitrário de funções pode estar no meio de sua execução em um determinado momento, mesmo que apenas uma esteja realmente sendo executada, isso indica que o contexto de várias funções deve ser mantido enquanto as mesmas não concluíram sua execução.

Fragmentação

Em computação, significa o desperdício de espaço disponível em memória.
Pode ser de dois tipos:

- Interna: Ocorre quando o processo não ocupa inteiramente os blocos de memória (páginas) reservados para ele. Geralmente acontece pois o tamanho do processo não é um múltiplo do tamanho da página de memória, o que acarreta

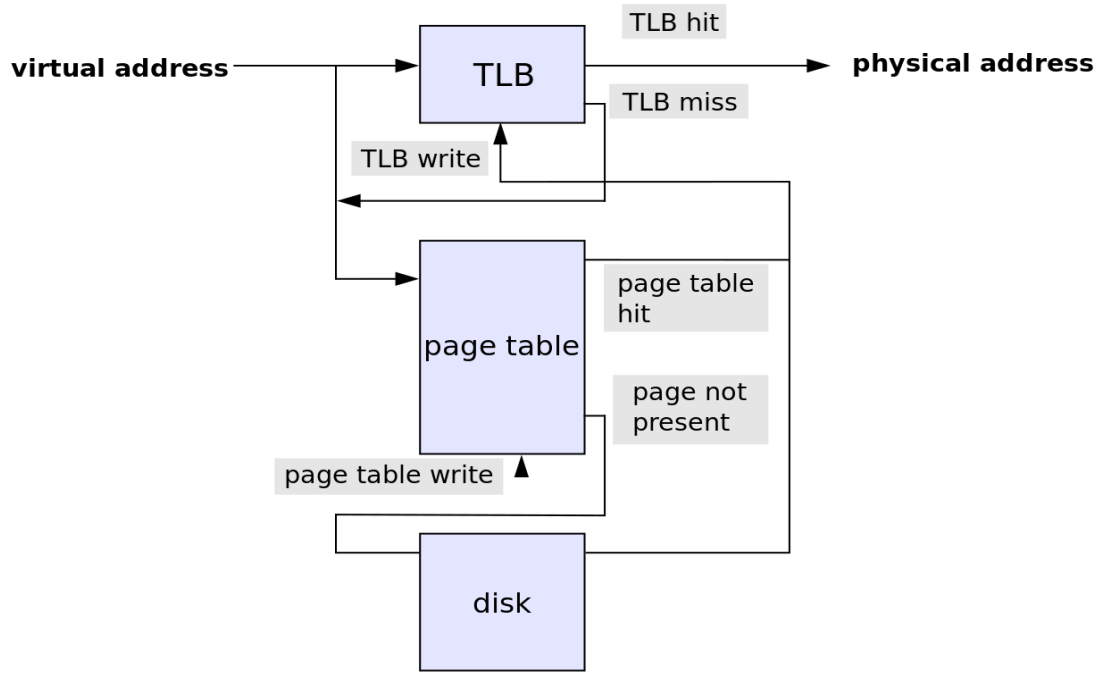
sobra de espaço na última página alocada. (dentro de um processo)
- Externa: Ocorre à medida que os programas vão terminando e deixando lacunas cada vez menores de espaços entre as páginas. Dependendo do tamanho que precisa ser escrito em memória, estes espaços podem ser pequenos demais para serem úteis, e assim ficam inutilizados. (entre processos)

Estratégias para "atacar" o problema com o algoritmos First-fit, Best-fit, Worst-fit e Next-fit.

Paginação

No contexto dos sistemas operacionais, a paginação da memória do computador é um processo de virtualização da memória que consiste na subdivisão da memória física em pequenas partições (frames), para permitir uma utilização mais eficiente da mesma. A alocação de memória é requisitada por páginas, a menor unidade deste método. Cada página é mapeada num frame de memória através de um processo chamado de paginação. O sistema operacional pode estar em base do espaço de endereçamento, em RAM, ou estar no topo do espaço de endereçamento, em ROM, e o restante do sistema mais embaixo, em RAM. O primeiro modelo foi inicialmente empregado em computadores de grande porte e mini-computadores (mas não é muito usado). O segundo modelo é utilizado em alguns computadores de mão e em sistemas embarcados. O terceiro modelo fez parte dos primeiros computadores pessoais, nos quais a parte do sistema contida em ROM é denominada BIOS. Quando o sistema é organizado dessa maneira, somente um processo pode ser executado a cada instante. Tão logo um usuário tecle um comando, o sistema operacional carrega o programa solicitado, do disco, para a memória e o executa. Quando o processo finaliza, o SO coloca na tela um caractere de prompt e espera por um novo comando. Ao receber um novo comando, carregará o novo programa na

memória, no espaço de endereçamento ocupado pelo programa anterior.



Relacionamento ente TLB e tabela de páginas para tradução de endereços virtuais em físicos

Translation Lookaside Buffer

A Translation Lookaside Buffer (TLB) é um conjunto de registradores especiais que são bastante rápidos. Ela está presente na Unidade de Gerenciamento de Memória de uma processador. Cada registrador tem duas partes: chave e valor. Dada uma chave, busca-se o valor correspondente. Geralmente o número de entradas não passa de 256[1] e a busca é feita em todos os registradores simultaneamente.

Algoritmos de Substituição de Página

🔍Ver artigo principal: Algoritmo de troca de página

- Algoritmo Ótimo

- Algoritmo Não Usada Recentemente
- Algoritmo FIFO
- Algoritmo Segunda Chance
- Algoritmo do relógio
- Menos Recentemente Usada
- WSClock

Em modelos de gerenciamento manual, podem ocorrer os problemas conhecidos como vazamento de memória, que acontece quando uma quantidade de memória é alocada e não é liberada ainda que nunca seja utilizada. Isto ocorre quando objetos perdem a referência sem terem sido liberados, mantendo o uso do espaço de memória.

Garbage Collector

É o gerenciamento automático de memória, também conhecido como coletores, sendo conhecido em Portugal como reciclagem automática de memória. Este serviço libera os blocos de memória que não sejam mais usados por um programa automaticamente. É oposto ao gerenciamento de memória manual, a alocação explicita e a desalocação dos recursos de memória do computador.
As vantagens desse tipo de gerenciamento são:

- Liberdade do programador: Não é obrigado ficar atento aos detalhes da memória;
- Menos bugs de gerenciamento de memória: Por se tratar de uma técnica mais confiável;
- Gerenciamento automático: Mais eficiente que o manual.

E entre as desvantagens, podemos citar:

- O desenvolvedor tende a estar mais desatento em relação a detalhes de memória;
- O gerenciador automático ainda apresenta limitações.

Quando deixam de existir referências a um objeto, este passa a ser considerado apto a ser "coletado" pelo garbage collector,

que significa dizer que será removido da memória, deixando-a livre para uso por outros objetos.

Os algoritmos de garbage collection operam de um modo que permite classificá-los em duas grandes famílias:

- Identificação Direta: por contagem de referências (reference counting);
- Identificação Indireta: por varrimento (tracing), que pode incluir também compactação da memória livre; cópia; ou geracional (utilizado nas máquinas virtuais Java e .NET).

Gerenciamento de memória no DOS

O IBM PC original foi projetado com uma memória RAM de 1024KB

- 640KB
  - Para o sistema operacional e programas
- 384KB - área de memória superior (Upper Memory Area) ou UMA
  - Para os adaptadores diversos como EGA & VGA, MDA, adaptadores CGA, e de redes.
  - ROM BIOS e Shadow RAM.
  - E mais tarde, área de paginação de expansão de memória (EMS) vista mais adiante.

Logo depois, foi provado que esta quantidade de memória se tornaria insuficiente para as necessidades futuras.
Entretanto, os sistemas operacionais e aplicativos desenvolvidos até então não seriam capazes de reconhecer um endereço de memória superior aos 640KB originais, o que levou os projetistas a desenvolverem ferramentas que executariam esta tarefa.

Emm386.exe

É o dispositivo de instalação da memória expandida (Expanded Memory) ou EMS. A EMS consistia em toda a memória acima dos 1024KB (1MB) original, em computadores baseados nas

tecnologias dos processadores 80286, 80386, i486 ou Pentium. A capacidade máxima de endereçamento fica limitada a 32MB e seu acesso é através de uma paginação de 64KB na área UMA. Os programas deveriam ser escritos de forma a poderem reconhecer a área EMS.
O nome "EMM" vem do inglês Extended Memory Manager.

Himem.sys

É o dispositivo de instalação da área de memória alta (High Memory Area), conhecida também como HMA. Sua principal função é controlar o uso da memória estendida do computador, de modo que dois ou mais aplicativos ou dispositivos não utilizem o mesmo endereço de memória ao mesmo tempo. Para as primeiras versões do Windows, o Himem.sys fornecia suporte para que este fosse executado em modo protegido.
O nome Himem vem do inglês High Memory.

Smartdrv.exe

É o gerenciador de memória cache de disco (no Novell DOS 7, é chamado de Nwcache.exe). Ambos possuem a mesma função que é a de minimizar o acesso ao disco, carregando um bloco de informações na memória para processamento, ao invés de ir buscar aos poucos no disco. Existiram também placas de expansão chamadas de Disk Accelerator, que possuem um hardware próprio para esta tarefa, como por exemplo o 'Disk Accelerator PROMISE DC4030VL. Atualmente, esta técnica é desempenhada por memórias localizadas na placa principal do sistema (cache on-board) resultando, portanto, dois modos de gerenciamento de memória cache de disco: por software e por hardware.
O nome Smartdrv é uma abreviação do inglês Smart Drive.

Multiprogramação com Partições Fixas

É usada em sistemas embarcados simples. Muitos dos programas modernos permitem que múltiplos processos executem

simultaneamente, ou seja, quando um processo for bloqueado, outro poderá usar a CPU aumentando a sua utilização. O melhor jeito de realizar a multiprogramação consiste em dividir a memória em n partições de tamanhos diferentes que podem ser criadas manualmente ao iniciar o sistema.

Ao chegar no sistema, um job é colocado em uma fila de entrada juntamente associada à menor partição existente, porém que seja grande o suficiente para armazená-lo e, como o tamanho dessas devidas partições são fixas, todo espaço que não é usado pelo job na partição será perdido. Quando jobs estão chegando torna-se evidente quando a fila para uma grande partição está vazia, mas a fila para uma pequena partição está cheia, nesse caso os jobs pequenos terão que esperar para que a memória seja liberada, mesmo que exista memória disponível.

O modo correto é manter uma fila única, sempre que a partição se torna disponível, o job que se encontra mais próximo do início da fila e que caiba na partição pode ser nela executado. Para que não haja um total desperdício de grandes partições com jobs pequenos, o ideal seria pesquisar em toda a fila de entrada e alocar a partição disponível ao maior job que nela possa ser carregado. Para que os jobs pequenos possam também serem executados sem desperdiçar partições maiores, seria necessária a criação de ao menos uma partição pequena.

Modelagem de Multiprogramação.

Tem o objetivo de melhorar o desempenho da CPU, se um processo permanecer em execução apenas 20% do tempo em que ocupa a memória, com cinco processos simultaneamente na memória a CPU deveria permanecer ocupada nesse intervalo de tempo, esse modelo não é realista porque os cinco processos nunca poderão esperar E/S ao mesmo tempo. Outro modelo é supor que um processo gaste uma fração p de seu tempo esperando pela finalização de sua solicitação de E/S, e com os n processos

executando ao mesmo tempo em memória, a probabilidade dos processos estarem aguardando E/S é p^n.

A probabilidade presume como independentes todos os n processos, que sendo assim é aceitável existirem 5 processos em memória, sendo que três deles executando simultaneamente e dois em estado de espera, mas, como se tem uma única CPU, não pode haver três processos executando ao mesmo tempo, de modo que se um processo estiver pronto para a execução, terá de esperar enquanto a CPU estiver ocupada com outro processo, portanto observa-se que os processos não são independentes.

Suponha que um computador tenha 32MB de memória, e um SO que use 16 MB e cada programa empregando 4MB. Com esses tamanhos possibilitam que os programas estejam simultaneamente na memória, considerando que um processo passa 80% de seu tempo em espera por E/S, o cálculo de utilização da CPU se dá por 1-(p^n), sendo, nesse caso, n=4 (pois a memória restante para os programas é de 16MB e cada programa tem 4MB, então 16/4) e o p=0,80, portanto tem-se uma utilização da CPU de 1-0,8^4 (1-p^n), aproximadamente 60%. Se houver a adição de mais 16MB permite que aumente seu grau de multiprogramação, aumentara a utilização da CPU para aproximadamente 83%. Ainda adicionando mais 16 MB, de 83% a utilização da CPU vai para aproximadamente 93%. O modelo permite que o dono do computador decida que a primeira adição de memória é um bom investimento, mas não a segunda que só aumentou 12% de sua utilização. Um modelo mais realístico pode ser baseado na teoria das filas.

Análise de Desempenho de Sistemas de Multiprogramação

O ultimo modelo usado em modelagem de multiprogramação pode ser usado para analisar sistemas em lote, por exemplo, digamos que um centro onde jobs gastariam 80% do seu tempo com espera

de E/S, em um certo dia quatro desses jobs são submetidos, o primeiro chega às dez horas, requer quatro minutos de tempo da CPU, considerando que os 80% do tempo é gasto com espera de E/S, o job usará apenas 12 segundos de tempo da CPU para cada minuto que estiver rodando na memória, mesmo que não haja outros jobs competindo com ele pela CPU.

Os demais 48 segundos serão gastos esperando pela conclusão de E/S, então o job terá de ficar na memória no mínimo 20 segundos para que possa obter quatro minutos de trabalho da CPU, mesmo sem competição de processos. Desde 10h às 10h10, o primeiro job consegue 2 minutos de trabalho da CPU. Quando o segundo job chega às 10h10, a utilização da CPU aumenta de 0,20 para 0,36.

No entanto, com o escalonamento round-robin (alternância circular), cada job consegue metade do tempo da CPU, obtendo 18 minutos de trabalho da CPU realizado para cada minuto de permanência na memória, a entrada do segundo job custa ao primeiro job somente 10% de perda em seu desempenho. Ele passa de 0,20 minutos de CPU para 0,18 para cada minuto de permanência em memória. Às 10h15 chega o terceiro job, até esse momento o job 1 tinha recebido 2,9 minutos de CPU e o segundo job 0,9 minutos de CPU. Com uma multiprogramação de grau três, cada job consegue 0,16 minuto de CPU por minuto de permanência na memória. Das 10h15 às 10h20, cada um dos três jobs consegue 0,8 minuto de tempo de CPU. Às 10h20 chega um quarto job e completa a sequência dos eventos.

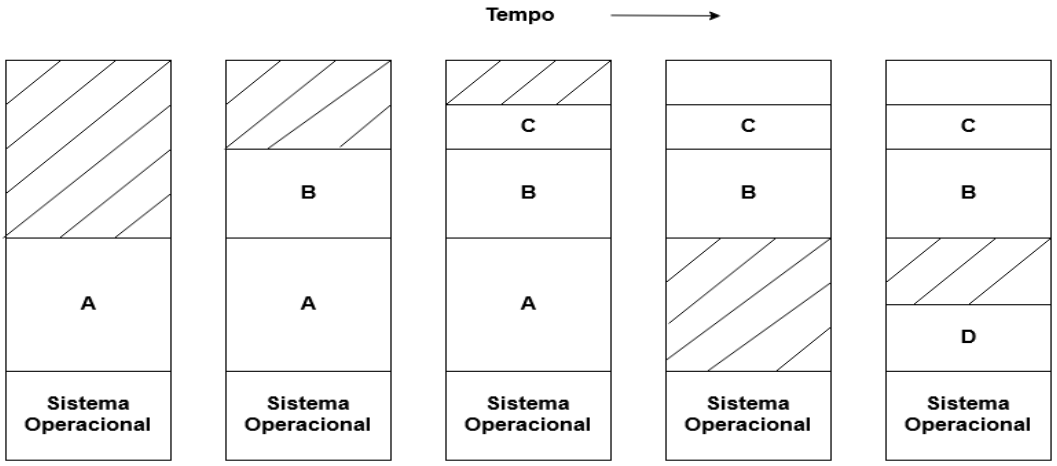
Troca de Processos

Em sistemas em lote, a memória é organizada em partições fixas. Em cada partição cada job ou processo é carregado ao alcançar o inicio da fila permanecendo até a conclusão de sua execução. Para garantir que a CPU esteja ocupada todo tempo é executado um determinado número de jobs, assim não necessitando utilizar uma outra técnica mais complicada.

Há uma diferença em sistemas com compartilhamento de tempo ou computadores gráficos pessoais, pois nesses casos pode ocorrer é insuficiente a quantidade memória principal para todos os processos ativos, sendo necessário armazenar o excedente em discos e trazidos dinamicamente para a memória quando precisarem ser executados.

Troca de Processos e Memória Virtual

São dois métodos usados para gerenciamento de memória utilizados conforme os recursos de hardware disponíveis:
Troca de Processos (também chamado swapping)



Um exemplo simples de Swapping, vemos o processo A ser executado e então devolvido ao disco.

Esse método trabalha trazendo cada processo para memória, e executa durante um tempo determinado e então devolve ao disco.

- Alocação de espaço para uma área de dados em expansão.
- Alocação de espaço para uma pilha e uma área de dados, ambos em expansão.

Transferir e remover processos da memória RAM constantemente irá inevitavelmente causar múltiplos espaços vazios ao longo do tempo. É possível combinar todos esses espaços em um espaço único, ao custo de muito tempo de processamento. Essa técnica é conhecida como compactação de memória.

Swapping tem uma grande desvantagem, a velocidade de transferência do disco é lenta. Por exemplo, aplicações pesadas como Photoshop demorariam vários segundos para serem carregados e removidos da memória RAM, pois a operação é limitada pelo dispositivo mais lento.

Memória Virtual
Já esse método permite que programas sejam executados mesmo que estejam apenas parcialmente carregados na memória principal.

Partições fixas e Partições variáveis: As principais diferenças são o tamanho e a localização das partições que variam conforme os processos entram e saem da memória nas partições variáveis, enquanto que nas partições fixas os parâmetros são fixos. Nas trocas de processos quando deixam muitos espaços vazios na memória, há a possibilidade aglutiná-los em único espaço contíguo de memória, movendo-os o máximo possível para os endereços mais baixos.

Técnica denominada compactação de memória. No entanto não é muito utilizada pelo tempo de processamento necessário considerado alto. Algo que deve ser dado uma devida importância é a quantidade de memória que deve ser alocada a um processo, quando for criado ou trazido do disco para memória. Se o processo possuir tamanho fixo, inalterável, então o processo de alocação torna-se simples: o sistema operacional alocará o espaço necessário. No entanto, na área de dados que o processo puder crescer, é alocado uma área temporária denominada (heap).

Se houver espaço disponível ao processo, ele poderá ser alocado a esse determinado processo. Quando os processos puderem ter duas área de expansão, a área de dados sendo usada como área temporária (heap) para variáveis dinamicamente

alocadas e liberadas, e uma área de pilha para variáveis locais e para endereços de retorno.

Multiprogramação com partições variáveis

O espaço de endereçamento de cada processo precisa ser contíguo (i.e. contínuo) para que se possa implementar o mecanismo de proteção descrito acima usando os registradores base e limite.

A tendência é de que a memória apresente vários espaços vazios (buracos) ao longo do tempo. Compactação de memória: lento (ainda que em memória RAM). A quantidade de memória exigida por um processo pode crescer durante a sua execução:

- Malloc = memória "heap".
- Pilha do programa = ao executar funções, os endereços de retorno e as variáveis locais são armazenadas na pilha do programa.

O SO reserva espaço extra para expansão a cada processo ao seu carregado na memória. Se o espaço previsto for insuficiente:

- O processo é abortado;
- O processo é deslocado para outro espaço livre maior;
- Outro processo é enviado ao disco para liberar a memória para o processo originário da demanda.

Gerenciamento de memória com Mapa de bits

O SO mantém 1 bit para indicar se cada bloco da memória (ou unidade de alocação) está ocupado (1) ou livre (0). A memória é dividida em unidades de alocação.
Considerações sobre o tamanho do bloco de memória:

- Quanto menor a unidade de alocação, maior será o mapa de bits.
- Pequeno: necessidade de muitos bits ⇒ uso ineficiente da memória. Exemplo: se tamanho do bloco = 1 byte, 1/9 da memória será utilizado para o mapa de bits.
- Grande: memória sub-utilizada, pois se o tamanho do processo não for múltiplo do tamanho da unidade de

alocação, uma quantidade de memória considerável será desperdiçada no último bloco.

Vantagens do uso de mapa de bits:

- simplicidade: o tamanho do mapa depende apenas do tamanho da memória e das unidades de alocação.

Desvantagens:

- Quando um processo necessita de k unidades de alocação, o gerenciador de memória deve encontrar uma sequência de k bits 0, o que se constitui um processo lento.

Gerenciamento com Listas Encadeadas

É mantida uma lista encadeada de segmentos alocados e vazios, sendo que cada segmento é um processo ou um buraco entre dois processos. A lista apresenta-se em ordem de endereços, e quando um processo termina ou é enviado para o disco, e a atualização da lista ocorre da seguinte maneira: cada processo, desde que não seja nem o primeiro nem o último da lista, apresenta-se cercado por dois segmentos, que podem ser buracos ou outros processos, o que nos dá as quatro possibilidades . O SO mantém uma lista ligada para indicar os segmentos de memória (sequência de blocos) livres (L) ou ocupados (P).

Cada nó contém os seguintes campos:

- Segmento de memória livre (L) ou ocupado (P);
- Início do segmento;
- Tamanho do segmento (em número de blocos);
- Próximo segmento.

A lista ligada pode estar ordenada pelo campo início do segmento (vantagem da atualização rápida da lista quando um processo termina):

- AXB ⇒ A_B
- AX_ ⇒ A__
- _XB ⇒ __B
- _X_ ⇒ ___

Os buracos adjacentes devem ser combinados num único. Para escolher o ponto em que deve ser carregado um processo recém criado ou que veio do disco por uma troca, vamos utilizar alguns algoritmos assumindo que o gerenciador de memória sabe quanto espaço alocar no processo:

- First Fit (primeiro encaixe): percorrer a fila até encontrar o primeiro espaço em que caiba o processo. É um algoritmo rápido.
- Next Fit (próximo encaixe): o mesmo que o algoritmo anterior, só que ao invés de procurar sempre a partir do início da lista, procura a partir do último ponto em que encontrou. Desempenho próximo ao anterior.
- Best Fit (melhor encaixe): consiste em verificar toda a lista e procurar o buraco que tiver espaço mais próximo das necessidades do processo. É mais lento, e desempenho pior que o First Fit.
- Worst Fit (pior ajuste): pega sempre o maior buraco disponível. Desempenho ruim.

Esses algoritmos podem ter sua velocidade aumentada pela manutenção de duas listas separadas, uma para processos e outra para buracos. Quando temos duas listas separadas, outro algoritmo é possível. É o Quick Fit (ajuste rápido), que consiste em manter listas separadas para alguns dos tamanhos mais comuns especificados (ex. uma fila para 2k, outra para 4k, outra para 8k etc). Neste caso, a busca de um buraco com o tamanho requerido, é extremamente rápido, entretanto, quando um processo termina, a liberação de seu espaço é complicada, devido à necessidade de reagrupar os buracos e modificá-los de fila.

COMUNICAÇÃO ENTRE PROCESSOS (IPC)

A comunicação entre processos, em inglês Inter-Process Communication (IPC), é o grupo de mecanismos que permite aos processos transferirem informação entre si.
A execução de um processo pressupõe por parte do sistema operativo, entre outras coisas, a criação de um contexto de execução próprio que, de certa forma, abstrai o processo dos componentes reais do sistema. Devido a esta virtualização dos recursos, o processo não tem conhecimento acerca dos outros processos e, como tal, não consegue trocar informação.

Modelos de Comunicação entre Processos

Memória Compartilhada: uma região de memória é compartilhada entre os processos.
Troca de Mensagem: Ocorre por meio de troca de mensagens entre os processos.
Ambiente de Cooperação entre Processos
• Motivos:
– Compartilhamento de informações
– Agilidade na computação
– Modularidade
– Conveniência
• Necessidade de mecanismo para prover a comunicação entre processos
– (Interprocess Comunication – IPC)
• Dois modelos:
– Memória compartilhada
– Troca de mensagem

Método	Sistemas Operativos
Sinais	Todos
Pipes unidireccionais	POSIX
Pipes nomeados	POSIX

Memória partilhada	POSIX
Filas de mensagens	POSIX
Remote Procedure Calls	Todos

Mecanismos locais

Pipes nomeados (FIFO)
Os pipes nomeados, ou FIFOs, são ficheiros especiais que servem de canal de comunicação entre processos. Um processo abre o FIFO para escrita, outro para leitura.

Pipes unidirecionais
Esta é a forma mais divulgada de IPC. Um exemplo:
C:\> type text.txt | more

Este exemplo em DOS iria canalizar o output do comando TYPE como input para o programa MORE.

Filas de mensagens
Uma fila de mensagens ou message queue permite criar uma zona de intercâmbio de mensagens de tamanho fixo ao nível do sistema operativo, oferecendo um maior nível de segurança sobre quais os processos que a ela podem aceder. Além do sistema de permissões, a fila de mensagens é criada com uma chave que deverá ser apenas do conhecimento dos utilizadores deste recurso.

Uma das característica deste mecanismo é que as mensagens podem ser retiradas selectivamente, já que cada uma é identificada por um tipo de mensagem. A extração pode ser por tipo ou do tipo FIFO.

Memória Compartilhada

Dois ou mais processos utilizam a região de memória compartilhada, conectando-a no seu espaço de endereçamento.
Deve se ter a garantia de que os dois processos não estejam gravando dados no mesmo local simultaneamente.

• Exemplo: Problema Produtor-Consumidor
• Paradigma para processos em cooperação • Processo produtor produz informações que são consumidas por um processo consumidor – Buffer ilimitado não impõe limite prático sobre o tamanho do buffer – Buffer limitado assume que existe um tamanho de buffer fixo.

```
#ifndef BUFFER_H_
#define BUFFER_H_
class Buffer {
public:
    int BUFFER_SIZE = 5;
    int count; // Número de itens no buffer.
    int in; // Posição de inserção, próxima posição livre.
    int out; // Próxima posição cheia.
    int *buffer;
    Buffer();
    virtual ~Buffer();
    void insert(int item);
    int remove();
    void print();
};
#endif /* BUFFER_H_ */
```

Buffer vinculado – método insert()

```
void Buffer::insert(int item) {
    while(count == BUFFER_SIZE) // Cheio?
        ; // faça nada.
    // add um item no buffer.
    count++;
    buffer[in] = item;
    // Incremento Circular.
    in = (in + 1) % BUFFER_SIZE;
}
```

Buffer vinculado – método remove()

```
int Buffer::remove() {
    int item;
    while(count == 0) // Vazio?
```

```
        ; // faça nada.
    // remove o item do buffer.
    count--;
    item = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
    return item;
}
```

Mecanismos cliente/servidor

Sockets
Os sockets também são considerados IPC, embora mais orientados para uma arquitectura cliente-servidor.

Remote Procedure Calls
Os RPC são também considerados IPC.

Sincronização entre processos

A sincronização entre processos permite gerir o acesso concorrente a recursos do sistema operativo de forma controlada por parte dos processos, de maneira que um recurso não seja modificado em simultâneo, ou que os processos não fiquem em espera que o recurso seja libertado.

Sinais
A sinalização é um mecanismo largamente utilizado em UNIX e funciona analogamente a um **trigger** (disparo). Um processo receptor de um sinal irá parar a sua execução imediatamente, para passar a processar o sinal. Desta forma, o processo é assim "despertado" para um qualquer evento, consoante o sinal recebido. Um exemplo comum é o sinal **KILL** (matar) enviado a um processo bloqueado:
kill -KILL 3516

Semáforos

Os semáforos são o mecanismo de sincronização mais complexo, já que permitem, simultaneamente, gerir o acesso concorrente quer em modo de exclusividade (1 utilizador) quer em modo de cooperação (N utilizadores).

Em modo de exclusividade o semáforo apenas permite um utilizador do recurso. A parte do código do processo delimitada pela activação do semáforo e sua libertação denomina-se secção crítica. Esta deverá ser o mais pequena e rápida possível, a fim de minimizar o tempo de espera dos processos concorrentes.

Este mecanismo é bloqueante, em oposição aos sinais.
Um dos primeiros mecanismos de comunicação interprocessos disponíveis em sistemas UNIX. • São interrupções de software que notificam ao processo que um evento ocorreu, são utilizados pelo núcleo.

- Permitem somente o envio de uma palavra de dados (código do sinal (1 a 64)) para outros processos.
- Não permitem que processos especifiquem dados para trocar com outros processos.
- Dependem do SO e das interrupções geradas por software suportadas pelo processador do sistema.
- Quando ocorre um sinal, o SO determina qual processo deve receber o sinal e como esse processo responderá ao sinal.

Quando sinais são gerados
- Criados pelo núcleo em resposta a interrupções e exceções, os sinais são enviados a um processo ou thread.
- Em consequência da execução de uma instrução (como falha de segmentação)
- Em um outro processo (como quando um processo encerra outro) ou em um evento assíncrono.

Sinais POSIX
Estão definidos 64 sinais.

1) SIGHUP 2) SIGINT 3) SIGQUIT 4) SIGILL 5) SIGTRAP 6) SIGABRT 7) SIGBUS 8) SIGFPE 9) SIGKILL 10) SIGUSR1 11) SIGSEGV 12) SIGUSR2 13) SIGPIPE 14) SIGALRM 15) SIGTERM 16) SIGSTKFLT 17) SIGCHLD 18) SIGCONT 19) SIGSTOP 20) SIGTSTP 21) SIGTTIN 22) SIGTTOU 23) SIGURG 24) SIGXCPU 25) SIGXFSZ 26) SIGVTALRM 27) SIGPROF 28) SIGWINCH 29) SIGIO 30) SIGPWR 31) SIGSYS 34) SIGRTMIN 35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3 38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8 43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7 58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2 63) SIGRTMAX-1 64) SIGRTMAX

- Um kill -9 pid tem o mesmo efeito de um kill -SIGKILL pid

Tratamento de Sinais

Processos podem:

– Capturar: especificando uma rotina que o SO chama quando entrega o sinal.

– Ignorar: Neste caso depende da ação padrão do SO para tratar o sinal.

– Mascarar um sinal: Quando um processo mascara um sinal de um tipo específico, o SO não transmite mais sinais daquele tipo para o processo até que ele desbloqueie a máscara do sinal.

<u>Um processo/thread pode tratar um sinal</u>

1. Capturando o sinal

– quando um processo capta um sinal, chama o tratador para responder ao sinal.

2. Ignorando o sinal

– os processos podem ignorar todos, exceto os sinais SIGSTOP e SIGKILL.

3. Executando a ação default

– Ação definida pelo núcleo para esse sinal.

- **Ações default**

– Abortar: terminar imediatamente.
– Descarga de memória: copia o contexto de execução antes de sair para um arquivo do núcelo (memory dump).
– Ignorar.
– Parar (isto é, suspender).
– Continuar (isto é, retomar a execução).

Bloqueio de Sinais

• Um processo ou thread pode bloquear um sinal.
• O sinal não é entregue até que o processo/thread pare de bloqueá-lo.
• Enquanto o tratador de sinal estiver em execução, os sinais desse tipo são bloqueados por default.
• Ainda é possível receber sinais de um tipo diferente (não bloqueados).
• Os sinais comuns não são enfileirados.
• Os sinais de tempo real podem ser enfileirados.

Mecanismos de passagem de mensagem

• Pipe:
– Permite a criação de filas de processos;
– Saída de um processo é a entrada de outro;
– Existe enquanto o processo existir; • Fifo (Named pipe):
– Extensão de pipe;
– Continua existindo mesmo depois que o processo terminar;
– Criado com chamadas de sistemas;
• O processo produtor escreve dados para o pipe.
• Depois disso, o processo consumidor lê dados do pipe na ordem “primeiro a entrar, primeiro a sair”.
• Quando um pipe é criado, um inode que aponta para o buffer do pipe (página de dados) é criado.
• O acesso ao pipe é controlado por descritores de arquivo.
– Podem ser passados entre processos relacionados (por exemplo, pai e filho).

- Pipes nomeados (FIFOs).
  - Podem ser acessados por meio da árvore de diretório.
- Limitação: buffer de tamanho fixo.

```
int pipe(int pipefd[2]);
int pipe2(int pipefd[2], int flags);
pipe() cria um pipe, um canal de dados unidirecional que pode
ser usado em IPC.
O array pipefd é usado para retornar dois descritores de
arquivo:
- pipefd[0] referencia o lado de leitura
- pipefd[1] o lado de escrita.
Os dados escritos são colocados em um buffer pelo kernel até
ser lido pelo lado de leitura.
Se flags é 0, então pipe2() é o mesmo que pipe().
Mais detalhes: man pipe
```

Exemplo de Pipe

```c
int main(void) {
    pid_t pid;
    int mypipe[2];
    /* Criar o pipe. */
    if (pipe(mypipe)) {
        fprintf(stderr, "Falha ao criar o Pipe.\n");
        return EXIT_FAILURE;
    }
    /* Criar o processo filho. */
    pid = fork();
    if (pid == (pid_t) 0) {
        /* No processo filho. */
        close(mypipe[1]);
        read_from_pipe(mypipe[0]);
        return EXIT_SUCCESS;
    } else if (pid < (pid_t) 0) {
        /* pid negativo, falha no fork. */
        fprintf(stderr, "Falha ao executar o Fork.\n");
        return EXIT_FAILURE;
    } else {
        /* Processo pai. */
        close(mypipe[0]);
        write_to_pipe(mypipe[1]);
        return EXIT_SUCCESS;
    }
}
```

Sockets

- Permitem que pares de processos troquem dados estabelecendo canais diretos de comunicação bidirecional.
- Usados principalmente para comunicação bidirecional entre vários processos em sistemas diferentes, mas podem ser usados para processos no mesmo sistema.
- Armazenados internamente como arquivos.
- O nome do arquivo é usado como endereço do socket, que é acessado por meio do VFS.

• **Sockets de fluxo**
– Implementam o tradicional modelo cliente/servidor.
– Os dados são transferidos como um fluxo de bytes.
– Usam TCP para comunicação, de modo que são mais apropriados quando a comunicação tem de ser confiável.
• **Sockets de datagrama**
– Comunicação mais rápida, mas menos confiável.
– Os dados são transferidos por meio de pacotes de datagramas.
• **Par de Sockets**
– Par de soquetes conectados e não denominados.
– Limitado para ser usado por processos que compartilham descritores de arquivo.

Passagem de mensagem

- Provê troca de mensagens entre processos rodando em máquinas diferentes;
- Utiliza-se de duas primitivas de chamadas de sistema: send e receive;
- Podem ser implementadas como procedimentos: send (destination,&message); receive (source,&message);
- O procedimento send envia para um determinado destino uma mensagem, enquanto que o procedimento receive recebe essa

mensagem em uma determinada fonte; Se nenhuma mensagem está disponível, o procedimento receive é bloqueado até que uma mensagem chegue.

Passagem de mensagem - Problemas

- Mensagens podem ser perder na transmissão;

– Mensagem especial acknowledgement (ack) o procedimento receive envia um ack para o procedimento send. Se esse ack não chega no procedimento send, esse procedimento retransmite a mensagem já enviada;

- A mensagem é recebida mas o ack se perde.

– receive checa se cada mensagem enviada pelo send satisfaz uma seqüência de números. Ao receber uma nova mensagem, receive verifica essa identificação, se ela for semelhante a de alguma mensagem já recebida, receive descarta a mensagem.

- Desempenho: copiar mensagens de um processo para o outro é mais lento do que operações com semáforos e monitores.
- Autenticação → Segurança;

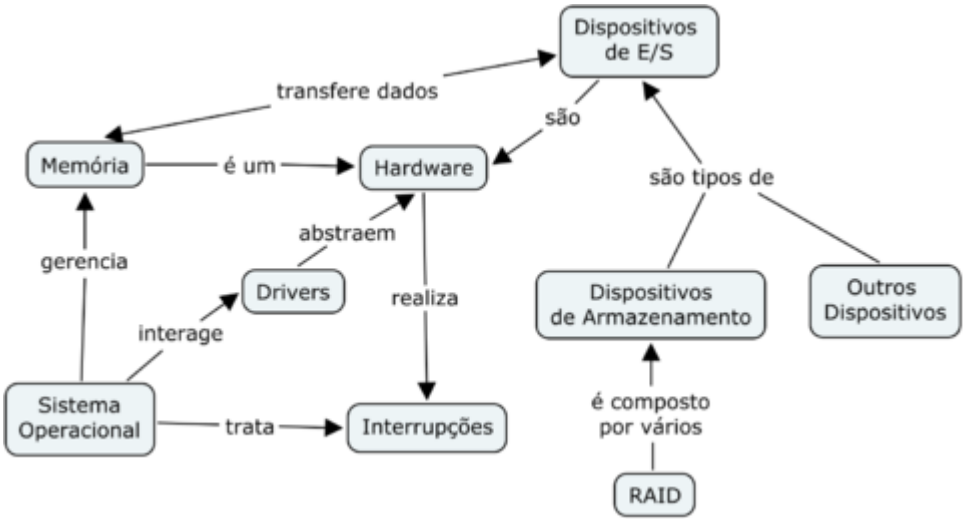
RMI

- Invocação de método remoto (RMI) é um mecanismo da Java semelhante às RPCs.
- RMI permite que um programa Java em uma máquina chame um método em um objeto remoto..

GERENCIAMENTO DE I/O(ENTRADA E SAÍDA)

Uma das principais funções do Sistema Operacional é gerenciar os dispositivos de Entrada e Saída (E/S) ligados ao computador.

É tarefa do sistema operacional enviar sinais, informando as ações que o usuário espera que o dispositivo realize; tratar as interrupções e erros gerados pelos dispositivos.



Existem duas visões sobre o hardware de E/S: A dos engenheiros, que os vêem como chips, ligações elétricas, etc. E a visão dos programadores, que vêem uma interface de programação para se comunicar com o dispositivo.

Controladores de dispositivos

Dispositivos de hardware precisam ser controlados para proporcionar a entrada e saída de dados para o processador. O controle do hardware é realizado por meio de hardware e software apropriados.

A porção de hardware é denominado controlador de hardware e segue padrões determinados pelo barramento (IDE, SCSI, USB, etc), assim, ligados a cada tipo de barramento existem controladores de hardware: controladora de hardware IDE, controladora de hardware SCSI, etc.

Para utilizar um dispositivo de hardware, é necessário conectá-lo a interface física da controladora de hardware. Por exemplo, um disco rígido IDE deve ser conectado a uma das quatro interfaces disponíveis pela controladora IDE. Em geral, o Sistema Operacional pode ter softwares controladores de dispositivo (driver de dispositivos). Os drivers de dispositivos para a controladora de hardware, geralmente são genéricos, embutidos no próprio Sistema Operacional. E os drivers para dispositivos de hardware são geralmente específicos, uma vez que controlam funcionalidades específicas providas pelos fabricantes.

Periférico é dado como qualquer dispositivo de hardware conectado a um computador de forma a permitir a sua interação com o mundo externo.

Princípios do software de E/S

Independência do dispositivo

Esse conceito trabalha sobre a possibilidade de escrever programas capazes de acessar um dispositivo E/S sem que seja necessário um conhecimento prévio sobre qual é o dispositivo. Portanto, um programa deverá ser capaz de ler/escrever um arquivo da mesma forma para qualquer dispositivo. Para o caso de se obter a entrada de um dispositivo para saída em outro - como em "/dispositivo1/arquivo > /dispositivo2", o Sistema Operacional fica incumbido de tratar dos problemas causados pelo fato de os dispositivos serem desiguais e necessitarem de sequências de comandos muito diferentes para leitura e escrita.

Nomeação uniforme

O nome de um arquivo ou de um dispositivo deveria ser uma cadeia de strings ou um número inteiro totalmente independente do dispositivo. Dessa forma, os arquivos e diretórios são endereçados pelo nome do caminho.

Tratamento de erros

De maneira geral, espera-se que os erros, como de leitura por exemplo, sejam tratados em níveis mais baixos, o mais próximo do hardware.

Tipos de conexão e de transferência de dados

Os dispositivos I/O podem se conectar de forma serial ou paralela. Na interface serial existe apenas uma linha por onde os dados trafegam. Na interface paralela os dados são transmitidos simultaneamente através das várias linhas para dados, a quantidade de linhas é um múltiplo de 1 byte (8 bits).

As informações sobre os endereçamentos de I/O ficam armazenadas, Sistema Operacional Linux, no arquivo /proc/ioports

A transferência pode ser síncrona (bloqueante) - após um read, o programa é suspenso até que os dados estejam disponíveis no buffer, ou assíncrona (orientada à interrupção) - a CPU inicia uma transferência e

segue realizando outra atividade até ser sinalizada por um interrupção (o que acontece na maioria das E/S físicas).

Software de E/S

O principal objetivo do software gerenciador de E/S é padronizar ao máximo o acesso e controle dos dispositivos, permitindo a inserção de novos dispositivos no sistema computacional sem a necessidade de um outro software auxiliar. Isso se torna uma tarefa bastante complicada devido à grande variedade, complexidade e particularidades dos dispositivos periféricos encontrados. Para facilitar isso, o software de E/S é geralmente é dividido em camadas. Cada camada tem uma função bem definida para executar e uma interface também bem definida para as camadas adjacentes, por meio de uma serie de operações comuns a todos os dispositivos. Uma forma de implementação dessa estrutura é dividir o software em quatro camadas, onde temos a camada superior sendo a E/S vista pelo usuário; a segunda camada o software que enxerga E/S da mesma forma, independente do dispositivo; a terceira camada serve como interface padrão para drivers, e a ultima (mais inferior), composta pelos drivers propriamente ditos.

Utilização de buffer

O buffer pode ser utilizado em momentos em que os dados não podem ser armazenadas em seu destino final - como acontece com os pacotes que são recebidos pela rede e que precisam ser examinados. Outro exemplo são os dispositivos que apresentam restrições de tempo real, em que os dados devem ser antecipadamente colocados em um buffer de saída a fim de separar a taxa com a qual o buffer é preenchido.Essa taxa é calculada a partir da taxa com a qual ele é esvaziado. Dessa forma, evita-se a sobreposição do buffer.

Dispositivos compartilhados vs dedicados

Alguns dispositivos, como discos, podem ser usados por vários usuários simultaneamente. Outros, como dispositivos de fita, devem ser dedicados a um usuário até que este termine sua tarefas. O Sistema Operacional deve ser capaz de tratar ambos, de forma a evitar problemas.

Impasses (Deadlock)

Como dito anteriormente, alguns dispositivos devem ser dedicados a um usuário até que este termine sua tarefa, não podendo ser interrompido para atender a solicitação de outro processo. Quando dois processos

alocam recursos para si de forma que nenhum dos dois possa realizar a tarefa, mas também nenhum dos dois disponibilizam estes recursos antes de realizar a tarefa estes processos encontram-se em um impasse (deadlock) e permaneceram ali até que um fator externo os retire dessa situação. O princípio básico do impasse é descrito formalmente: Um conjunto de processos está em um impasse se cada processo do conjunto está esperando um evento que somente outro processo do conjunto pode causar. Como todos os processo estão esperando, nenhum deles jamais causará qualquer dos eventos que poderiam acordar qualquer dos outros membros do conjunto e todos os processos continuam a esperar eternamente.

Dispositivos de Entrada e Saída

Dispositivos de entrada e saída também chamados de dispositivos de I/O e são classificados em três tipos: Caractere, Bloco, Pseudo-dispositivos.

Essa classificação não é 100% abrangente, uma vez que relógios (temporizadores) em hardware não podem ser classificados em nenhum destes tipos.

Tais dispositivos se interagem com o SO através de interrupções que podem ser tratadas pelo proprio SO para que seus dados de I/O sejam encaminhadas corretamente.

Nem todos os dispositivos de E/S se enquadram nessas duas categorias, por exemplo o relógio que apenas gera sinais de interrupção. Tal dispositivo trabalha em uma ampla variação de velocidade, o que impõe uma considerável pressão sobre o software, que deve trabalhar bem diante das mais diferentes ordens de magnitude de velocidade.

Dispositivos de Caractere

Os conhecidos dispositivos do tipo *dispositivos de caracteres* são assim chamados por terem sua comunicação feita através do envio e recebimento de um fluxo de caracteres. São usados, muitas vezes, para comunicação com dispositivos de interface serial. Geralmente as implementações desse tipo priorizam a eficiência da comunicação em vez do seu volume, neste sentido, é feita de forma não "bufferizada", sendo assim cada caracter é lido/escrito no dispositivo imediatamente [necessita citação]. A exemplo de dispositivos de caracteres que priorizam a comunicação e não necessitam de buffer temos o teclado e o mouse.A exemplo de dispositivos de caracteres que já priorizam o volume ao invés da comunicação temos a impressora, devido a quantidade de dados

que chega ser bem maior que sua velocidade de impressão, logo possui um buffer de impressão.

Dispositivo de caracteres: É aquele que envia/recebe um fluxo de caracteres, sem considerar qualquer estrutura de blocos, eles não são endereçáveis e não dispõe de qualquer operação de posicionamento. Impressoras e mouses são exemplos desses dispositivos.

Dispositivos de Bloco

Os dispositivos do tipo *dispositivos de bloco* são similares aos dispositivos do tipo caractere, porém com uma diferença: O modo de transmissão dos dados, que é feita na forma de blocos. São dispositivos que a comunicação é feita por meio de blocos de dados como em HD's, drivers de CD-ROM, flash drivers e dispositivos de armazenamento em geral.

Outra grande diferença é que os dispositivos de bloco em geral utilizam operações de entrada/saídas *bufferizadas*, no sentido de otimizar o desempenho da transferência de dados. O Sistema Operacional aloca um buffer em um tipo de memória para transferir blocos para cada processo de Entrada e Saída (E/S). Quando um processo envia/requisita dados para/de um dispositivo, o buffer deve ser preenchido para concluir a operação de E/S. Ao encher, o buffer completo é transferido e esvaziado para que seja liberado para uma nova operação.

Dispositivo de blocos é aquele que armazena informação em bloco de tamanho fixo, cada um com seu próprio endereço. O tamanho dos blocos normalmente variam de 512 bytes a 32 K bytes. A propriedade essencial de um dispositivo de blocos é que cada bloco pode ser lido/escrito independente dos outros. Discos, são exemplos desses dispositivos.

Pseudo-Dispositivos

Em sistemas do tipo UNIX, arquivos de dispositivo especiais podem não possuir um dispositivo físico correspondente, estes são chamados de "pseudo-dispositivos".

Ex: /dev/null (recebe dados a serem descartados, como uma 'lixeira') , /dev/random (gera números aleatórios) , /dev/zero (gera valores com valor 'zero', útil para criar arquivos preenchidos com esse valor).

Interrupção

Pedido de Interrupção (IRQ)

O Sistema Operacional (SO) chaveia entre os aplicativos ativos para que o usuário tenha a sensação de que estes estão executando em paralelo. O SO permite que um aplicativo utilize o processador durante um determinado período de tempo e então permite que outra aplicação o utilize. Como o chaveamento é feito de uma forma muito rápida temos a impressão de que os aplicativos estão sendo executados ao mesmo tempo.

Um pedido de interrupção (abreviação IRQ (em inglês)) é a forma pela qual componentes de hardware requisitam tempo computacional da CPU. Um IRQ é a sinalização de um pedido de interrupção de hardware.

Por exemplo: caracteres digitados no teclado, operações de leitura sobre o HD, dados recebidos pelo modem ou mesmo movimentos do mouse devem ser executados mesmo que a máquina esteja processando alguma tarefa.

Dessa forma IRQ´s são canais de comunicação com o processador. Ao receber um pedido através de algum destes canais, o processador percebe a solicitação de interrompimento de um dispositivo.

Quando um programa de usuário emite uma chamada ao sistema, esta é encaminhada ao driver apropriado. Para evitar que a CPU fique ocupada interrogando se dispositivo terminou a operação de E/S (**espera ociosa**), o driver pede ao dispositivo que lhe sinalize quando isso ocorrer. Dessa forma, o Sistema Operacional poderá executar outras tarefas enquanto o programa que o chamou pedindo o serviço se encontra bloqueado. Ao final da operação, o controlador do dispositivo gera uma interrupção, ao chip controlador de interrupção, para sinalizar à CPU.

Caso nenhuma outra interrupção esteja pendente ou em tratamento e nenhum outro dispositivo fez uma requisição de maior prioridade no mesmo momento, a interrupção é tratada imediatamente. Caso contrário, ela é ignorada, e o dispositivo continuará emitindo sinal de interrupção.

Após executar uma instrução, a CPU verifica as linhas de interrupções para saber se alguma interrupção foi sinalizada

Caso tenha sido sinalizada uma interrupção, o hardware grava os registradores da CPU na pilha do programa que estava em execução e carrega o contador de programa com o endereço da rotina referente à interrupção sinalizada.

Interrupções podem ser geradas por hardware ou por software. No caso do hardware, pelo dispositivos periféricos ou pelo relógio (timer). As interrupções geradas por software são conhecidas como **system calls** (chamadas ao sistema) ou **trap** (armadilha). A diferença primordial entre as interrupções geradas por software e as geradas por hardware, está no fato de que, conhecendo um programa e seus dados, é possível prever quando as interrupções de software irão acontecer - o que não é possível no caso das interrupções de hardware.

Interrupções revisitadas

Em hardware, as interrupções trabalham da seguinte forma: quando um dispositivo de E/S finaliza seu trabalho ele gera uma interrupção, enviando um sinal pela linha do barramento à qual está associado. O sinal é detectado pelo chip controlador de interrupção localizado na placa mãe, o qual decide o que fazer. Caso esse chip esteja em uso por outro sinal de interrupção ou seja enviado para esse chip outro sinal de maior prioridade, o sinal é posto em uma fila para quando o chip ficar ocioso ele tratar esse sinal.

O hardware sempre salva certas informações antes de iniciar a rotina de tratamento da interrupção; no mínimo, o contador de programas (PC) deve ser guardado.

Existem duas categorias de interrupções:

- Interrupção precisa → Deixa a máquina em um estado bem definido. Possui 4 propriedades:
    - 1. O contador de programa é salvo em um lugar conhecido;
    - 2. Todas as instruções anteriores àquela apontada pelo PC foram totalmente executadas;
    - 3. Nenhuma instrução posterior à apontada pelo PC foi executada;
    - 4. O estado de execução da instrução apontada pelo PC é conhecido.
- Interrupção imprecisa → Deixa a máquina em um estado não bem definido, pois tem concorrência interna. Complica bastante a vida do projetista do SO, que se vê então obrigado a calcular o que já foi executado e aquilo que ainda não foi executado, para isso é necessário guardar muito mais informação que o normal.

Tratadores de interrupção

As interrupções deveriam ser escondidas para que a menor parte possível do SO soubesse de sua existência. A melhor maneira é bloquear o driver que iniciou uma operação de e/s até que a e/s se complete e uma interrupção ocorra.

O efeito resultante da interrupção fará com que o driver previamente bloqueado esteja novamente apto a executar. Esse modelo funciona bem sempre que os drivers são estruturados como processos do núcleo do sistema operacional, com seus próprios estados, suas pilhas e seus contadores de programa.

Interrupções em sistemas Linux

Em antigos computadores, existiam apenas 8 interrupções de hardware. Nos atuais, existem 16 tipos de interrupções de hardware, numeradas de 0 a 15.

As interrupções que ocorrem em computadores Linux podem ser vistas digitando no shell o comando:

$ cat /proc/interrupts

0 = É o timer do SO.

1 = Interrupção de teclado.

2 = Funciona como ponte para a interrupção 9, e é advinda dos antigos computadores com apenas 8 IRQ's. Dessa forma, ela permite que esses computadores sejam utilizáveis, quando for necessário uso da interrupção 8 para cima.

3 = geralmente usada pela ttyS1, podendo ser usada por outros dispositivos como placas de rede.

4 = geralmente usada pela ttyS0. Quando é usado um mouse serial, ele trabalho através dessa IRQ.

5 = É a segunda porta paralela. Como muitos computadores não possuem essa porta, ela é muito usada pela placa de som.

6 = Usada pelo controlador de disquete

7 = Antigamente, era a primeira porta da impressora. Atualmente várias impressoras não usam IRQ's.

8 = Relógio

9 = Ponte com a IRQ2.

10 = É uma interrupção geralmente livre. Geralmente usada por controladores USB.

11 = Outra interrupção livre

12 = Mais uma IRQ livre. Quando há um mouse PS/2, é trabalhada essa IRQ

13 = Processador de dados numérico.

14 = Usada pela primeira controladora de discos rígidos.

15 = Usada pela segunda controladora de discos rígidos.

Convém citar que as IRQ's 14 e 15 não podem ser compartilhadas, devido ao alto grau de importância das mesmas.

Os dispositivos PCI/PCI Express são feitos para permitir o compartilhamento de interrupções. Esse trabalho de compartilhamento pode ser feito pelo chipset.

Uso de memória por dispositivos

E/S mapeada na memória

Cada controlador tem alguns registradores usados para a comunicação com a CPU. Por meio da escrita nesses registradores, o SO pode comandar o dispositivo para entregar ou aceitar dados, alternar em ligar/desligar, ou ainda executar alguma outra tarefa. A partir da leitura desses registradores o SO pode descobrir o estado do dispositivo, se ele está preparado para aceitar um novo comando, etc. Além dos registradores, usualmente os dispositivos também possuem um buffer.

Existe duas formas para a CPU ter acesso a esses registradores e buffers:

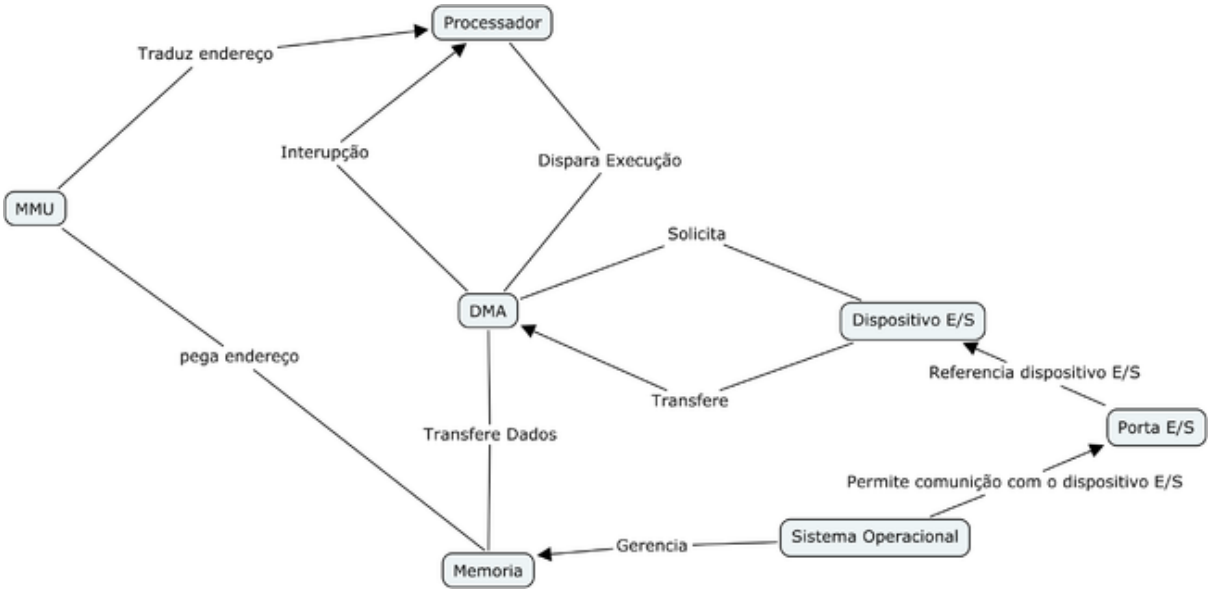
- 1º → Cada registrador de controle é associado a um número de porta de E/S, usando uma instrução especial de E/S (IN REG, PORT OUT REG, PORT) que seria um acesso direto.
- 2º → Mapear todos os registradores de controle no espaço de endereçamento da memória. Cada registrador de controle é associado a um endereço de memória único ao qual nenhuma memória é associada, que seria o mapeamento da E/S na memória.

Essa segunda forma, nos traz duas grandes vantagens, a primeira é que o programador pode tratar esses registradores como se trata a memória normal, isso é, como uma variável, isso faz com que um driver possa ser completamente escrito em C/C++, caso seja feita a primeira abordagem, necessariamente o driver terá algum código em ASSEMBLY. A outra vantagem é que não é necessário qualquer mecanismo de proteção especial para impedir que os processos do usuário executem E/S, tudo o que o SO

tem de fazer é deixar de mapear aquela porção do espaço de endereçamento associada aos registradores de controle no espaço de endereçamento virtual do usuário.

Acesso Direto à Memória (DMA)

O Acesso Direto à Memória (DMA) é uma das técnicas utilizadas para otimizar o uso de memória por dispositivos. O DMA é um componente de hardware que permite a transferência direta de dados entre dispositivos periféricos e a memória principal, tornando assim dispensável a participação da CPU neste processo.



O controlador de DMA é um hardware desenvolvido para desempenhar toda a sequência de transferência de dados acessando diretamente a memória. Ele gerencia vários canais que podem ser programados para realizar a transferência de dados, quer seja de um dispositivo para a memória ou vice-versa.

O SO somente pode usar o DMA se o hardware tem o controlador de DMA. O DMA melhora o desempenho do sistema, pois poupa tempo ocioso da CPU, que poderia muito bem executar a tarefa do DMA, porém como o tempo de E/S é grande principalmente para grandes quantidades de dados pode fazer com que a CPU fique muito tempo ociosa. Quando a quantidade é pequena as vezes é até mais viável fazer a transferência direto pela CPU que é um hardware mais rápido que o DMA, isso pode causar

concorrencia no barramento, pois o barramento utilizado pelo DMA para acessar a memória é o mesmo utilizado pela CPU. Utilizando o DMA, a CPU requisita ao DMA de onde começar a ler os bytes, quantos bytes devem ser lidos/escritos e fica livre para executar outras tarefas que sejam CPU Bound, então quando o DMA termina de realizar sua tarefa, ele transmite um sinal de interrupção para a CPU que simplesmente usa os bytes.

Note que a CPU pode fazer exatamente o que o DMA faz, isso fica a cargo de projeto. Coloca uma requisição de leitura no barramento e fica esperando até receber os bytes e assim poder usa-lo, a diferença é que usando a CPU para transferência de uma quantidade maior de dados, poderá ocasionar em CPU ociosa.

Um computador tem dois gerenciadores de DMA, divididos em canais. Os canais 0 a 3 são gerenciados por um gerenciador, enquanto os canais 4 a 7 vão para o outro.

Para visualizar quais canais estão em uso em sistemas Linux basta digitar no shell o comando:

$ cat /proc/dma

Os oito canais e seus usos são descritos a seguir: 0 = usado pelo refresh da memória Ram dinâmica. 1 = placas de som de 8 bits, adaptadores SCSI, placas de rede. 2 = controladora de disquete 3 = porta paralela ECP, placas de som de 8 bits, adaptadores SCSI, placas de rede, controladores de scanner antigos. 4 = ponte para a controladora de DMA 0~3 5 a 7 = placas de som, adaptadores SCSI, placas de rede.

Os canais 1 a 3 operam sob 8 ou 16 bits, enquanto os canais 5 a 7 operam apenas sob 16 bits.

Dispositivos PCI (e outros de alta velocidade) possuem um controlador de DMA embutido, muito mais rápido que o DMA simples. Por exemplo, esse controlador é usado em discos rígidos atuais e pode atingir velocidades de 66MB/s.

Processo de Entrada e Saída (E/S)

E/S programada

Neste método o processador executa o programa e tem o controle total sobre as operações de entrada e saída:

- I - Os dados são copiados para o núcleo;

- II - O Sistema Operacional envia, para a saída, um caractere de cada vez;
- III - A cada caractere enviado, a CPU verifica se o dispositivo está pronto para receber outro;

Esse comportamento é chamado de espera ociosa ou polling.

A desvantagem desse método é que como o processador geralmente é mais rápido que o dispositivo de E/S ocorrerá um desperdício de processamento.

E/S orientada à interrupção

Diferentemente da E/S programada, em que a CPU fica verificando o dispositivo para saber se ele está pronto para mais operações ou se terminou, na E/S orientada à interrupção, a CPU passa a realizar outras tarefas escalonadas até que seja informada pelo dispositivo, por meio de uma interrupção, que este está pronto para mais caracteres ou concluiu sua tarefa. Quando isso acontecer, o processador para o que está fazendo para executar o **tratador de interrupção**, quando sua execução estiver terminada, o processador volta à rotina que foi interrompida inicialmente.

Para que seja empregada essa política de interrupções existem detalhes de software e de hardware que devem ser atendidos, e para que esses detalhes sejam executados, a maioria dos computadores possuem um hardware denominado **controlador de interrupções**. As principais funções do controlador de interrupções são:

1. identificar a fonte da interrupção;
2. priorizar uma interrupção em relação a outra;
3. selecionar quais interrupções serão atendidas.

O emprego de interrupções libera o computador para realizar cálculos, então o processador fica responsável apenas por iniciar a operação de entrada-saída, e quando esta for concluída, executar o tratador de interrupção.

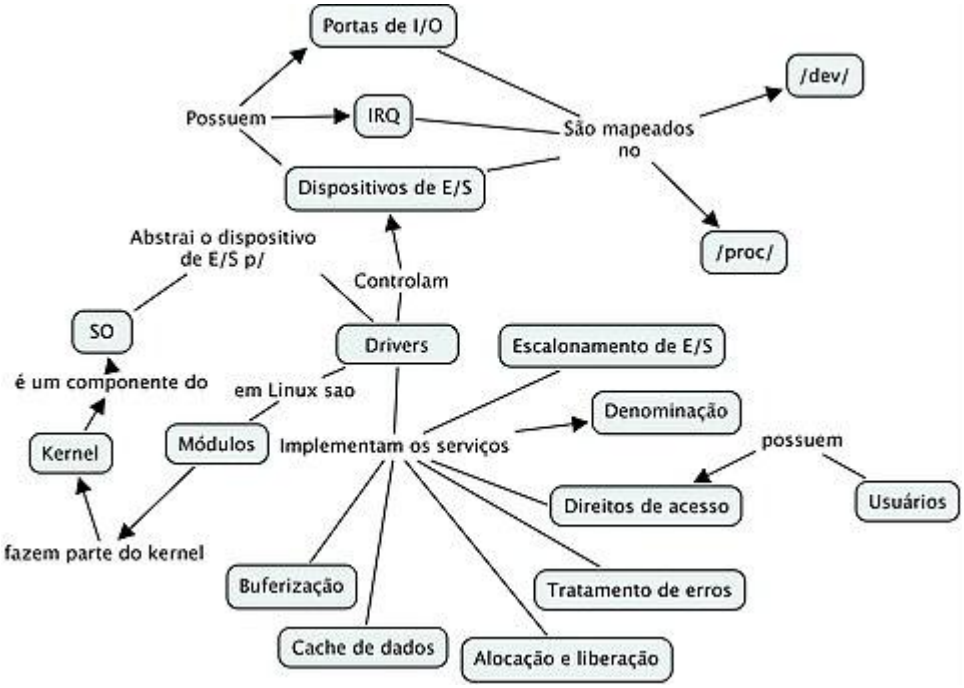
E/S usando DMA

No método de E/S orientada à interrupção, a cada caractere processado, é gerado uma nova interrupção à CPU. Para diminuir o peso de processamento sobre a CPU, que perderia muito tempo por conta das contínuas interrupções, passa-se tal tarefa para o DMA, que passará a administrar as interrupções por buffer (não mais por caractere). O DMA executa então a E/S programada (neste caso, não é a CPU que faz o trabalho, mas sim o controlador do DMA).

Quando existe uma quantidade de dados significante para ser transferida a técnica de Interrupções se torna ineficaz, sendo melhor a utilização de um hardware especial (**Controlador de DMA**), que transfere os dados diretamente de um dispositivo de E/S para a memória, ou vice e versa.

A transferência por DMA acontece quando o processador inicializa o controlador DMA, fornecendo todas as informações necessárias sobre os dados a serem transferidos (quantidade de dados, origem e destino dos blocos e ainda o sentido da transferência), depois ele dispara a execução do DMA e enquanto a transferência estiver ocorrendo o processador pode se dedicar a outra tarefa. Ao final da transferência o DMA sinaliza ao processador por meio de uma interrupção de hardware.

Drivers



O que são drivers

Um driver é uma camada de software que faz a comunicação do sistema operacional com o controlador do hardware que por sua vez faz a interface com o *hardware*. Drivers escondem as diferenças entre os diversos dispositivos, através de uma interface de programação única.

Driver de dispositivo é responsável por implementar as rotinas necessárias ao acesso e à gerência de um dispositivo específico.

A camada de drivers de dispositivo representa uma parte significativa do sistema de entrada e saída em relação às funcionalidades. Ela é responsável por implementar as rotinas necessárias ao acesso e à gerencia de um dispositivo específico.É necessário que o software de E/S realiza a programação de registradores internos de controladores que compõem a interface física dos dispositivos e implementa os respectivos tratadores de interrupção. Assim, cada tipo de dispositivo requer um driver apropriado. Essa camada fornece uma abstração a mais genérica possível para a camada superior, a de E/S independente do dispositivo.

Cada dispositivo de E/S ligado ao computador precisa de algum código específico do dispositivo para controlá-lo. Esse código, chamado de driver do dispositivo.

Para acessar o hardware do dispositivo, o driver normalmente deve ser parte do núcleo do SO.

Os sistemas operacionais geralmente classificam os drivers dentre algumas poucas categorias. As categorias mais comuns são dispositivos de bloco - os quais contêm vários blocos de dados que podem ser endereçados independentemente - e os dispositivos de caractere, os quais geram ou aceitam uma sequencia de caracteres.

A maioria dos SOs define uma interface-padrão para todos os drivers de blocos e uma segunda interface-padrão para todos os drivers de caracteres. Essas interfaces consistem em um número de procedimentos que o restante do so pode utilizar para fazer o driver trabalhar para ele.

Um driver de dispositivo apresenta várias funções. A mais óbvia é aceitar e executar requisições abstratas, de leitura ou gravação, de um software independente de um dispositivo localizado na camada acima da camada de drivers dos dispositivos. Mas existem também algumas poucas outras funções que ele tem de executar.

Drivers no Linux X Drivers no Windows

Enquanto no Windows, os drivers são desenvolvidos pelos próprios fabricantes do dispositivo, precisando ser instalados manualmente e seguindo de um processo de *reboot* do sistema. Em ambientes GNU/Linux, a instalação dos "drivers" são incorporados diretamente ao Kernel e vêm pré-instalados no sistema.

Drivers Linux

Em sistemas GNU/Linux os "drivers" são chamados de módulos. O kernel desse sistema é dito monolítico com vantagens de micro-kernel já que os sistemas GNU/Linux são LKM(Loadable Kernel Modules), já que os "drivers" ou módulos são carregáveis ao sistema sem a necessidade de um novo processo de *bootstrap* após a instalação. Para maior facilidade com o usuário, esse carregamento é feito no processo de *bootstrap*. Na verdade na maioria dos sistemas operacionais que contam com o desenvolvimento da comunidade do software livre, os módulos são desenvolvidos pela comunidade de desenvolvimento. Se esse módulo tem uma boa aceitação pela própria comunidade e passou por todos os requisitos propostos pela equipe de desenvolvimento do kernel do sistema, esse pode passar a fazer parte do próprio kernel do sistema. Dessa forma, o usuário não precisa correr atrás da instalação uma vez que o driver está incorporado e instalado no sistema.

Drivers Windows

Nem sempre a companhia que desenvolveu certo hardware, também tem que desenvolver o driver para o mesmo. Há casos em que o hardware foi desenvolvido sobre um certo padrão de hardware. Nesses casos, um driver genérico é desenvolvido pela Microsoft para esse dado padrão. Nem todos os drivers se comunicam diretamente com o dispositivo. Pode haver uma pilha de drivers para determinado dispositivo, em que parte deles age como um filtro, transformando os dados de um formato para o outro, enquanto apenas a base da pilha se comunica diretamente com o dispositivo. Isso pode ser melhor visualizado por uma imagem disponibilizada pela própria Microsoft em seu site:

http://i.msdn.microsoft.com/dynimg/IC535115.png

Também, nem sempre os drivers estarão associados a um dispositivo. Também existe os “Software Drivers”. Em determinados momentos, uma aplicação precisa de acessar recursos que ela só poderia acessar em modo kernel, porém ela está em modo usuário. Então, divide-se essa aplicação em dois componentes: a aplicação que rodará em modo usuário, que fará a interface com o usuário e um driver que rodará em modo kernel, dando acesso aos recursos necessários. Esse driver que roda em

modo kernel é chamado de Software Driver. No site da Microsoft também está disponível uma imagem que ilustra isso:

http://i.msdn.microsoft.com/dynimg/IC535116.png

Em casos em que determinado hardware desenvolvido não segue um padrão pré-existente, que tenha driver desenvolvido pela Microsoft, a companhia também precisa desenvolver o driver. No site da Microsoft, existem explicações sobre como construir um driver para Windows, dicas de performance e outros. Podemos encontrar essas informações em: http://msdn.microsoft.com/en-us/windows/hardware/gg454507

Para desenvolver esses drivers, o desenvolvedor deve saber conceitos do funcionamento interno do Windows, como o gerenciamento de memória, fluxo de entrada e saída, entre outros. Para ajudar em seu desenvolvimento, a Microsoft disponibiliza o Windows Driver Kit (WDK), com uma série de utilitários para criar um driver, como bibliotecas, ambiente de desenvolvimento, exemplos, etc. Como parte do WDK, também é disponibilizado o Windows Driver Foundation (WDF). O WDF define um único modelo de driver que pode ser usado para criar drivers orientado a objetos, tanto para o modo kernel quanto para o modo usuário. Ele também inclui frameworks para drivers de modo kernel e modo usuário, além de uma série de utilitários de verificação para o driver.

Formas de instalação

Em Windows, esses drivers são desenvolvidos pelos fabricantes do dispositivo. É criado um arquivo executável somente com o binário desse driver para a instalação e utilização. Geralmente é um arquivo *.exe. Esse processo faz com que o usuário fique totalmente alienado daquilo que está acontecendo com a sua máquina durante a instalação.

Em GNU/Linux, os módulos podem ser de vários tipos. O usuário pode baixar o código-fonte, compila-lo e instala-lo na sua máquina. Esse é um processo bastante dificultoso para quem não tem muito conhecimento técnico nesses sistemas. Também o usuário pode recorrer ao módulo

pré-compilado do sistema. Esse é um processo bastante similar ao do Windows já que é um script executável pelo gerenciador de pacotes do sistema da máquina.

Dispositivos de armazenamento

É um dispositivo capaz de gravar(armazenar) informação(dados). Essa gravação de dados pode ser feita virtualmente, usando qualquer forma de energia. Um dispositivo de armazenamento retém informação, processa informação, ou ambos. Um dispositivo que somente guarda informação é chamado de mídia de armazenamento. Dispositivos que processam informação podem tanto quanto acessar uma mídia de gravação podem tanto acessar uma mídia de gravação portátil, ou ter um componente que armazena dados.

Tipos de dispositivos de armazenamento

os dispositivos de armazenamento se diferem quanto ao tipo de mídia, tipo de armazenamento(volátil e não-volátil), capacidade de armazenamento e velocidade de escrita.

Armazenamento por meio magnético

Dentre os dispositivos de E/S, podemos considerar o disco magnético como sendo o mais importante, devido diversos papéis que ele desempenha junto ao Sistema Operacional. Dentre esses papéis estão:

- Armazenamento de dados;
- Armazenamento de dados com tolerância às falhas de segurança de dados (RAID);
- Utilização como memória virtual (auxiliando a memória RAM, quando esta não pode guardar todos as páginas de dados processos, etc);

Os discos magnéticos podem ser confeccionados em diferentes materiais (plástico, metal, etc), recobertos por uma parte em material magnético. Essa parte magnética será responsável pelo armazenamento de dados. Esse armazenamento de dados, dá-se pela escrita por parte de um dispositivo de Hardware (cabeçote de leitura e gravação), o mesmo incidirá um campo magnético que definirá de maneira binária a informação. A união dessas áreas definidas pelo cabeçote formará uma informação (dados).

Da mesma maneira que dados podem ser escritos no disco magnético, eles podem ser recuperados (pelo mesmo dispositivo de Hardware) que

localizaria os dados, então faria a leitura magnética do que foi anteriormente gravado.

Indo mais a fundo sobre as regiões magnéticas do disco, tem-se que ele é dividido em trilhas circulares concêntricas, separadas por "gaps" (que evita problemas de alinhamento entre as trilhas). Existem duas arquiteturas diferentes para essas trilhas:

- CAV - número constante de bits a cada trilha. ---
- CLV - número de bits por trilha, dependedente da posição da trilha (mais interna ou mais externa).

CAV: Constant Angular Velocity. CLV: Constant Linear Velocity.

Um aumento de temperatura é concorrente ao bom armazenamento de dados, pois ocorre a desorientação dos "spins" (binários). Esses efeitos devem ser prevenidos pelos desenvolvedores do disco.

Como dito anteriormente, existem diversos tipos de dispositivos de E/S, bem como diversos tipos de discos magnéticos. São muitas as tecnologias, envolvendo diferentes maneiras de leitura e escrita no mesmo, diferentes maneiras de se agrupar camadas de disco, diferentes maneiras de trabalhar a velocidade do giro disco, etc.

Armazenamento por meio óptico

Os tipos de armazenamento por meio óptico sâo comumente empregados para o armazenamento de informações multimídia, como por exemplo som e vídeo.

O funcionamento desses dispositivos se dá por meio de um feixe de laser de alta precisão, que é projetado na superfície da mídia, está por sua vez, possui sulcos microscópicos capazes de desviar o laser em diferentes direções, representando assim as informações armazenadas na forma de bits.

Exemplos de dispositivos de armazenamento ópticos são: CD-ROMs, CD-RWs, DVD-ROMs, DVD-RWs, etc.

Armazenamento por meio eletrônico

São dispositivos de memórias de estado sólido ou SSDs. Possuem memória não volátil e funcionam por meios de circuitos eletrônicos.

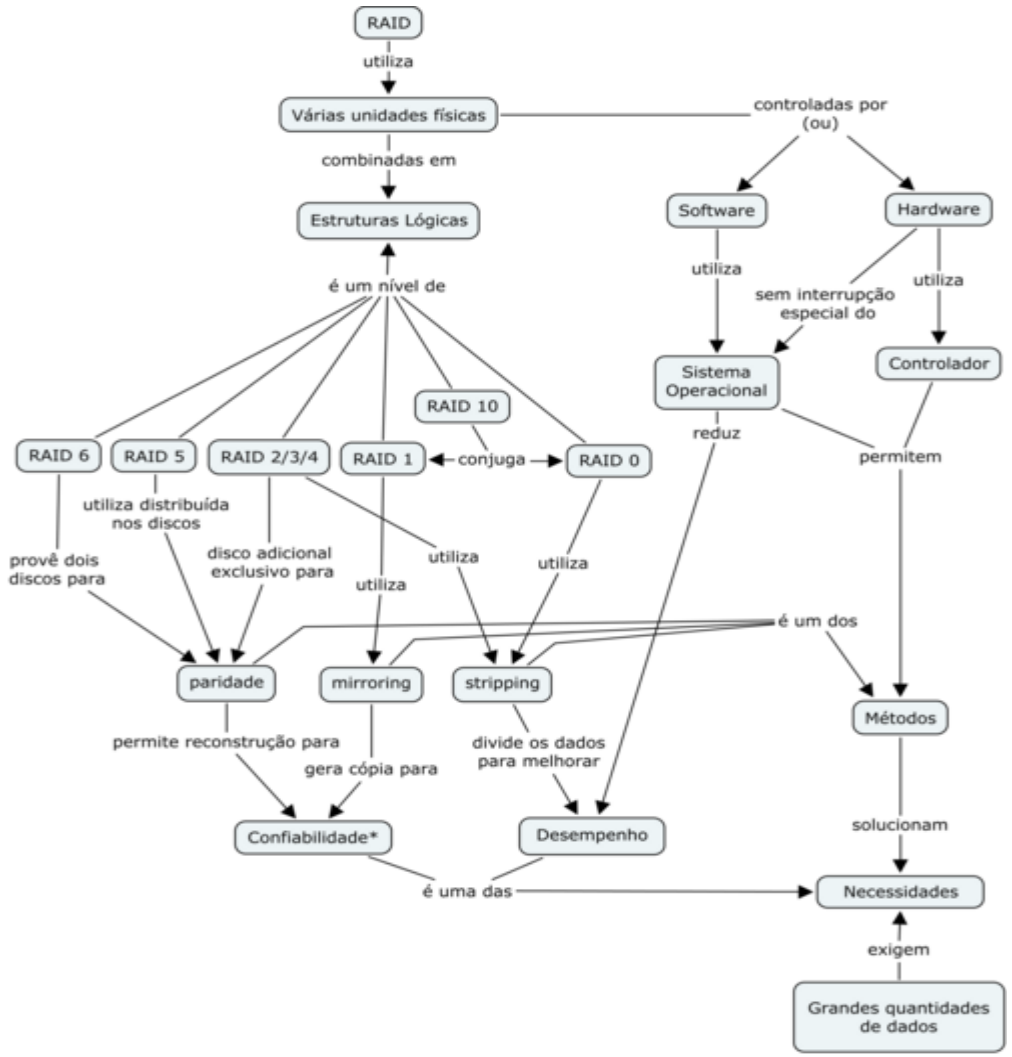
Encontramos dispositivos de armazenamento eletrônico em pen drives, cartões de memória e até mesmo em discos rígidos, onde são usados como buffer.

As gravações de dados em dispositivos de armazenamento eletrônico se da pelo carregamento de elétrons na camada de óxido existente entre duas portas feitas de materiais semicondutores, essas duas portas representam um dígito binário (bit). Assim, a presença de elétron entre essas duas portas indica que o ativamento do bit, ou seja, valor 1, caso não haja a presença de elétrons representa o bit inativo ou valor 0.

Outros dispositivos

RAID

A tecnologia RAID é utilizada para controlar o uso de várias unidades físicas (HDs) em um único servidor na tentativa de suprir duas de suas necessidades básicas, que são a confiabilidade e o desempenho. A partir deste ponto, alguns métodos (paridade, mirroring, stripping) são utilizados isolados ou combinados para compor diferentes estruturas lógicas (RAIDs 0, 1, 2, 3, 4, 5, 6, 10) que podem ser controladas por software ou por hardware. Vale a pena destacar que o primeiro caso, de controle por software, apresenta pior desempenho, uma vez que os métodos são realizados diretamente pelo Sistema Operacional, enquanto que no segundo caso, controlado por hardware, o método utilizado é transparente para o SO e ocupa apenas o hardware, que necessita ser capaz de suportar tal técnica.



OS NÍVEIS DE RAID

RAID NÍVEL 0 - Também chamado de "Striping" ou "Fracionamento" este nível é usado quando se deseja melhorar a performance do computador, nele os dados são divididos em segmentos, que tem um tamanho específico de bloco, e divididos entre os discos proporcionando grande velocidade na gravação e leitura de informações pois os dados são gravados ao mesmo tempo nos discos. Logo quanto maior o número de discos, mais rápido é a distribuição dos dados. Esse nível não aceita falhas uma vez que não existe redundância ao gravar os dados, logo se um hd der problema pode ocasionar perda de informações.

RAID NÍVEL 1 - Também chamado de "Mirroring" ou "Espelhamento" este nível é usado quando se deseja garantir a integridade dos dados, nele os dados são clonados para outro dispositivo, ou seja, se um computador com dois disco rígidos utilizam este nível, ao gravar dados em um disco automaticamente será gravado no outro tornando-os cópias idênticas, a gravação se torna mais lenta uma vez que precisa ser gravado duas

vezes, entretanto a leitura é mais rápida pois existem duas fontes de dados idênticas. Se houver falha em um disco ou perda de dados basta recuperar o setor defeituoso copiando os arquivos contidos do outro disco, esse nível é muito usado em servidores de arquivos.

RAID NÍVEL 2 - Este nível é responsável por monitorar os discos quanto a falhas garantindo uma maior consistência nos dados caso ocorra falta de energia durante a escrita. Essa funcionalidade perdeu utilidade a partir do momento em que os fabricantes de disco rígidos passaram a implementar mecanismos internos para detectar falhas e garantir a integridade dos dados.

RAID NÍVEL 3 - Para utilizar esse nível pelo menos 3 discos são necessários, nesse nível os dados são divididos entre dois discos e o terceiro fica responsável pelas informações de paridade, assim todos os bytes tem sua paridade exceto 1 que identifica o erro, através dessas informações é possível identificar onde o erro ocorreu garantindo a integridade dos dados em caso de recuperação, a sua montagem é via software e exige que todos os eixos das unidades de discos estejam sincronizados.

RAID NÍVEL 4 - Utiliza a mesma técnica do nível 3, porém em caso de falhas os dados são reconstruídos em tempo real através da utilização da paridade calculada a partir dos outros discos, sendo que cada um pode ser acessado de forma independente. Se algum dos discos estragar a paridade é usada imediatamente para reconstruir o conteúdo e os outros discos, que armazenam dados, são configurados para utilizarem segmentos grandes permitindo leituras independentes da informação armazenada.

RAID NÍVEL 5 - Oferece mais desempenho em relação ao nível 4 pois agora a paridade encontra-se distribuída nos discos, fazendo com que a gravação seja mais rápida pois não é necessário acessar um único disco a cada gravação, esse é o nível mais utilizado.

RAID NÍVEL 6 - Semelhante ao nível 5 porém usa o dobro de bits de paridade garantindo a integridade de até dois disco rígidos falharem ao mesmo tempo, o RAID 6 pode ser utilizado para sistemas de missão-critica aonde a confiabilidade dos dados é essencial.

RAID NÍVEL 0+1 - É o nível mais caro de ser implementado pois exige no mínimo 4 discos rígidos, esse nível implementa rapidez e integridade dos dados pois ele combina os níveis 0 e 1 onde os dados são divididos entre os discos para melhorar o rendimento, mas também utilizam outros discos para duplicar as informações.

REDES DE COMPUTADORES

Rede de computadores ou **redes de dados**, na informática e na telecomunicação é um conjunto de dois ou mais dispositivos eletrônicos de computação (ou módulos processadores ou nós da rede) interligados por um sistema de comunicação digital (ou link de dados), guiados por um conjunto de regras (protocolo de rede) para compartilhar entre si informação, serviços e, recursos físicos e lógicos.[1] Estes podem ser do tipo: dados, impressoras, mensagens (e-mails), entre outros. As conexões podem ser estabelecidas usando mídia de cabo ou mídia sem fio.

Os dispositivos integrantes de uma rede de computadores, que roteiam e terminam os dados, são denominados de “nós de rede" (ponto de conexão), que podem incluir hosts, como: computadores pessoais, telefones, servidores, e também hardware de rede. Dois desses dispositivos podem ser ditos em “rede” quando um dispositivo é capaz de trocar informações com o outro dispositivo, quer eles tenham ou não uma conexão direta entre si.

Os exemplo mais comuns de redes de computadores, são: Internet; Intranet de uma empresa; rede local doméstica; entre outras.

Comunicação

O *sistema de comunicação* vai se constituir de um arranjo topológico, interligando os vários módulos processadores através de enlaces físicos (meios de transmissão ou rede de transmissão), e de um conjunto de regras com o fim de organizar a comunicação (protocolos).

A Internet é um amplo sistema de comunicação que conecta muitas redes de computadores. Existem várias formas e recursos de diversos equipamentos que podem ser interligados e

compartilhados, mediante meios de acesso, protocolos e requisitos de segurança.

Os meios de comunicação podem ser: linhas telefônicas, cabo, satélite ou comunicação sem fios (wireless).

> O objetivo das redes de computadores é permitir a troca de dados entre computadores e a partilha de recursos de hardware e software. [2].

Uma *rede de computadores* também é formada por um número ilimitado mas finito de módulos autônomos de processamento interconectados, no entanto, a independência dos vários módulos de processamento é preservada na sua tarefa de compartilhamento de recursos e troca de informações.

Não existe nesses sistemas a necessidade de um sistema operacional único, mas sim a cooperação entre os vários sistemas operacionais na realização das tarefas de compartilhamento de recursos e troca de informações.

> conectores RJ-45 usados para conectar redes ethernet em informática.

Aplicações

Aplicações comerciais

A necessidade de compartilhar entre as pessoas de uma empresa informações ou até mesmo equipamentos entre computadores,

coloca Redes de computadores em evidência. Quando falamos de aplicações comerciais a palavra chave é **compartilhamento de recursos**[3], todos os integrantes de uma rede de computadores comercial necessitam ter acesso aos dados e equipamentos independentemente de sua localização física e recursos computacionais. Um cenário clássico que descreve muito bem o compartilhamento de recursos é o de uma empresa em que cada funcionário possui uma impressora de porte pequeno individualmente, é evidente que uma ou mais impressoras de um porte maior compartilhado entre os empregados seria mais vantajoso que privatizar individualmente uma única unidade para cada um, visto que nem todos usam com a mesma frequência e logo não há a necessidade de se gastar tanto recuso financeiro para comprar e manter impressoras individuais.[3]

Na contemporaneidade, cada vez mais empresas tem dependência de dados computacionais, portanto mais importante que o compartilhamento de equipamentos talvez seja o de dados comerciais entre membros de uma companhia. Uma pane em uma rede de qualquer empresa seria um golpe fatal para o funcionamento dela, talvez menos em empresas menores mas com certeza causaria prejuízos tanto temporais quanto financeiros para qualquer uma e de qualquer porte.

O modelo de distribuição de dados nomeado servidor é bastante utilizado, os dados são mantidos em um computador e local isolado e controlado por um administrador de sistemas. Os clientes (usuários usando seu computador) através da rede local acessam essa máquina chamada servidor, e com isso conseguem obter os dados que estão centralizados. Esse modelo exemplificado é chamado de modelo cliente-servidor e é bastante usado principalmente em aplicações Web.[3]

Aplicações domésticas

Hoje em dia diversos dispositivos eletrônicos domésticos estão integrados a **rede doméstica**, como lâmpadas inteligentes, sistema de câmeras de segurança, dispositivos de som, computadores, impressoras entre outros onde os usuários conectados à rede podem acessar e controlar os dispositivos,

ou também os usuários podem realizar o compartilhamento de informações entre si.

O acesso doméstico a internet também possibilita que os usuários possam acessar computadores (servidores) remotamente para fins de comércios eletrônicos, se comunicar com outras pessoas através de redes sociais, contratar serviços, ler jornais publicados online, comprar livros disponibilizados digitalmente. O usuário pode também pesquisar por qualquer coisa que lhe interesse como: notícias e informações sobre política, famosos, saúde, entretenimento e educação. Pode também utilizar da rede para o seu lazer como: jogos, receitas culinárias, esportes, e filmes/séries, entre outras infinitas possibilidades de conteúdos e atividades que podem ser encontradas pela rede mundial de computadores (**World Wide Web**). [4]

Propriedades

A **rede de computadores** pode-se dizer que é um ramo de engenharia elétrica, engenharia eletrônica, informática, tecnologia da informação(TI), telecomunicações ou engenharia da informática. Uma rede de computadores facilita as comunicações interpessoais permitindo que os usuários se comuniquem de forma eficaz e de maneira simples através de vários meios: e-mail, mensagens instantâneas, chat online, telefone e videoconferência.

Uma rede permite o compartilhamento de recursos de rede e computação. Os usuários podem acessar e usar recursos fornecidos por dispositivos na rede, como imprimir um documento em uma impressora de rede compartilhada, ou usar um dispositivo de armazenamento compartilhado. Também permite o compartilhamento de arquivos, dados e outros tipos de informações que dão aos usuários autorizados a capacidade de acessar informações armazenadas em outros computadores na rede.

Uma rede de computadores pode ser utilizada por hackers de segurança para implantar vírus de computador ou Worms[5] de computadores em dispositivos conectados à rede, ou para evitar que esses dispositivos acessem a rede através de um ataque de negação de serviço.[6]

Pacote de rede

Os links de comunicação por computador que não suportam pacotes, como os links tradicionais de telecomunicações ponto a ponto, simplesmente transmitem dados como um fluxo de bits. No entanto, a maioria das informações em redes de computadores é transportada em pacotes. Um pacote de rede é uma unidade de dados formatada (uma lista de bits ou bytes, normalmente algumas dezenas de bytes com alguns quilobytes de comprimento) carregados por uma rede comutada por pacotes. Os pacotes são enviados através da rede para o seu destino. Uma vez que os pacotes chegam, eles são remontados em sua mensagem original.

Os pacotes consistem em dois tipos de dados: informações de controle e dados do usuário (carga útil). As informações de controle fornecem dados que a rede precisa fornecer os dados do usuário, por exemplo: endereços de rede de origem e de destino, códigos de detecção de erros e informações de seqüência. Normalmente, as informações de controle são encontradas em cabeçalhos de pacotes e reboques, com dados de carga útil entre eles.

Com os pacotes, a largura de banda do meio de transmissão pode ser melhor compartilhada entre os usuários do que se a rede fosse comutada por circuito. Quando um usuário não está enviando pacotes, o link pode ser preenchido com pacotes de outros usuários e, portanto, o custo pode ser compartilhado, com relativamente pouca interferência, desde que o link não seja usado demais. Muitas vezes, a rota que um pacote precisa passar por uma rede não está disponível imediatamente. Nesse

caso, o pacote está em fila e aguarda até que um link seja gratuito.

Classificação

- **Segundo a arquitetura de rede**:
    - Arcnet (Attached Resource Computer Network)
    - Ethernet
    - Token ring
    - FDDI (Fiber Distributed Data Interface)
    - ISDN (Integrated Service Digital Network)
    - Frame Relay
    - ATM (Asynchronous Transfer Mode)
    - X.25
    - DSL (Digital Subscriber Line)

- **Segundo a extensão geográfica**
    - SAN (Storage Area Network)
    - LAN (Local Area Network)
    - WLAN (Wireless Local Area Network)
    - PAN (Personal Area Network)
    - MAN (Metropolitan Area Network)
    - WMAN (Wireless Metropolitan Area Network), é uma rede sem fio de maior alcance em relação a WLAN
    - WAN (Wide Area Network)
    - WWAN (Wireless Wide Area Network)
    - RAN (Regional Area Network)
    - CAN (Campus Area Network)
    - HAN (Home Area Network)
- **Segundo a topologia**:
    - Rede em anel (Ring)
    - Rede em barramento (BUS)
    - Rede em estrela (Star)
    - Rede em malha (Mesh)
    - Rede em ponto a ponto (ad-hoc)

        - Rede em árvore
- **Segundo o meio de transmissão:**
    - Rede por cabo
        - Rede de cabo coaxial
        - Rede de cabo de fibra óptica
        - Rede de cabo de par trançado
    - Rede sem fios
        - Rede por infravermelhos
        - Rede por micro-ondas
        - Rede por rádio

Software de rede

O hardware foi o único personagem nas primeiras estruturações de redes, mas com o passar do tempo percebeu-se que o software poderia trazer novas viabilidades e hoje ele é um elemento essencial das Redes de Computadores.[4]

Hardware de rede

- Elementos de cabeamento:
    - Cabo coaxial
    - Cabo de fibra óptica
    - Cabo de par trançado
    - Repetidor
    - Transceptor
- Estação de trabalho
- Placa de rede
- Concentrador (*hub*)
- Comutador (*switch*)
- Roteador (*router/gateway*)
- Modem
- Porta de ligação (*gateway router*)
- Ponte (*bridge*)
- *Firewall*
- Servidor
    - Servidor de arquivos
    - Servidor de comunicações
    - Servidor de disco

        - Servidor de impressão
        - Servidor de bluetooth

Modelo OSI

- Camada física
    - modem
- Camada de enlace de dados
    - **Ethernet**
    - PPP
- Nível de Rede
    - **IP**
    - IPX
- Nível de transporte
    - TCP
    - UDP
- Nível de sessão
    - NetBIOS
    - IPX
    - Appletalk
- Nível de apresentação
- Nível de aplicação
    - SMTP
    - FTP
    - Telnet
    - SSH
    - IRC
    - HTTP
    - POP3
    - VFRAD

Normas

- IEEE 802
- X.25

Técnicas de transmissão

- Banda larga

- Banda base

Modelagem de rede de computadores

Uma rede pode ser definida por seu tamanho, topologia, meio físico e protocolo utilizado.

- **PAN** (*Personal Area Network, em português: Rede de Área Pessoal*): é uma rede doméstica que liga recursos diversos ao longo de uma residência. Em outras palavras, é uma rede de computadores usada para comunicação entre computador e diferentes dispositivos tecnológicos de informação perto de uma pessoa. Alguns exemplos de dispositivos que são usados em um PAN são computadores pessoais, impressoras, aparelhos de fax, telefones, PDAs, scanners e até mesmo consoles de videogames. Uma PAN pode incluir dispositivos com fio e sem fio. O alcance de uma PAN normalmente se estende a 10 metros. Uma PAN com fio geralmente é construído com conexões USB e FireWire enquanto tecnologias como Bluetooth e comunicação por infravermelho tipicamente formam um PAN sem fio.
- **LAN** (**Local Area Network**, ou Rede Local). É uma rede onde seu tamanho se limita a apenas uma pequena região física. Uma rede de área local (LAN) é uma rede que conecta computadores e dispositivos em uma área geográfica limitada, como uma casa, escola, prédio de escritórios ou grupo de edifícios bem posicionado. Cada computador ou dispositivo na rede é um nó. LANs com fio são geralmente baseadas em tecnologia Ethernet. Novos padrões como o ITU-T G.hn também fornecem uma maneira de criar uma LAN com fio usando a fiação existente, como cabos coaxiais, linhas telefônicas e linhas de energia. [26] Contudo, atualmente em locais onde realizar o cabeamento possa ser um trabalho custoso, a rede LAN pode ser projetada para utilizar o protocolo 802.11, ou seja, uma rede wireless (sem fio), porém é uma conexão que pode possuir um desempenho inferior em relação a redes cabeadas com tecnologia Ethernet, visto

que os sinais wireless normalmente terão que enfrentar obstáculos como paredes que podem interferir na transmissão dos sinais.

As características definidoras de uma LAN, em contraste com uma rede de área ampla (WAN), incluem maiores taxas de transferência de dados, alcance geográfico limitado e falta de dependência de linhas alugadas para fornecer conectividade. A Ethernet atual ou outras tecnologias LAN IEEE 802.3 funcionam a taxas de transferência de dados de até 100 Gbit / s, padronizadas pelo IEEE em 2010. [27] Atualmente, a Ethernet de 400 Gbit / s está sendo desenvolvida.

Uma LAN pode ser conectada a uma WAN usando um roteador.

- **VAN** (**Vertical Area Network**, ou rede de área vertical). É usualmente utilizada em redes prediais, vista a necessidade de uma distribuição vertical dos pontos de rede.
- **CAN** (**Campus Area Network**, ou rede de área do campus). Uma rede que abrange uma área mais ampla, onde pode-se conter vários prédios dentro de um espaço continuo ligados em rede. Esta segundo Tanenbaum em seu livro "Redes de computadores" é uma LAN, justamente porque esta área dita ampla, abrange 10 quarteirões ou aproximadamente 2.500m quadrados. Esta rede é pequena quando comparado a uma cidade.
- **MAN** (**Metropolitan Area Network**, ou rede metropolitana). A MAN é uma rede onde temos por exemplo: Uma rede de farmácias, em uma cidade, onde todas acessam uma base de dados comum. As MAN oferecem altas taxas de transmissão, baixas taxas de erros, e geralmente os canais de comunicação pertencem a uma empresa de de telecomunicações que aluga o serviço ao mercado. As redes metropolitanas são padronizadas internacionalmente pela IEEE 802 e ANSI, sendo que os padrões mais conhecidos para a construção MANs são o

DQDB (*Distrubuted Queue Dual BUS*) e o **FDDI** (*Fiber Distributed Data Interface*).

Um exemplo bem conhecido de MAN é a rede de TV a cabo, onde as centrais recebem o sinal de uma grande antena, normalmente instaladas em topos de colina para melhor recepção do sinal e, então, o sinal é conduzido por cabos até as residências. A rede de televisão foi sendo aprimorada até o final da década de 1990, sendo melhorada a qualidade de imagem e áudio, acesso a mais canais especializados, maior distribuição do sinal nas cidades, etc. Quando a Internet se tornou atrativo para grande parte da população, as operadoras de TV perceberam que com algumas mudanças, era possível entregar internet full-duplex, dessa forma se transformando em uma rede metropolitana. O padrão 802.16 é uma MAN conhecida como WiMAX, que é uma rede metropolitana que permite a conexão de computadores, porém sem fio.

- **WAN** (**Wide Area Network**, ou rede de longa distância). Uma WAN integra equipamentos em diversas localizações geográficas (hosts, computadores, routers/gateways, etc.), envolvendo diversos países e continentes (*Ex: a Internet, as redes dos bancos internacionais, como o Citibank*).
- **SAN** (**Storage Area Network**, ou Rede de armazenamento). Uma SAN serve de conexão de dispositivos de armazenamento remoto de computador para os servidores de forma a que os dispositivos aparecem como locais ligados ao sistema operacional.
- **VPN** (**Virtual Private Network**, ou Rede virtual privada). É uma rede de comunicação privada comumente utilizada por grandes empresas, que consiste em possuir um ou mais computadores com alta capacidade de armazenamento e de processamento chamado de servidor, onde os dados da empresa e suas filiais podem ficar armazenados. Assim de forma compartilhada filiais empresariais de diferentes estados, cidades ou países podem ter acesso às informações da empresa de maneira remota e ágil.

Topologia

A **topologia de rede** é o canal no qual o meio de rede está conectado aos computadores e outros componentes de uma **rede de computadores**. Essencialmente, é a estrutura topológica da rede, e pode ser descrito física ou logicamente. Há várias formas nas quais se podem organizar a interligação entre cada um dos nós (computadores) da rede. Existem duas categorias básicas de topologias de rede:

- Topologia física
- Topologia lógica

A topologia física é a verdadeira aparência ou layout da rede, enquanto que a lógica descreve o fluxo dos dados através da rede. A topologia física representa como as redes estão conectadas (layout físico) e o meio de conexão dos dispositivos de redes (nós ou nodos). A forma com que os cabos são conectados, e que genericamente chamamos de topologia da rede (física), influencia em diversos pontos considerados críticos, como a flexibilidade, velocidade e segurança.

A topologia lógica refere-se à maneira como os sinais agem sobre os meios de rede, ou a maneira como os dados são transmitidos através da rede a partir de um dispositivo para o outro sem ter em conta a interligação física dos dispositivos. Topologias lógicas são frequentemente associadas à Media Access Control, métodos e protocolos. Topologias lógicas são capazes de serem reconfiguradas dinamicamente por tipos especiais de equipamentos como roteadores e switches.

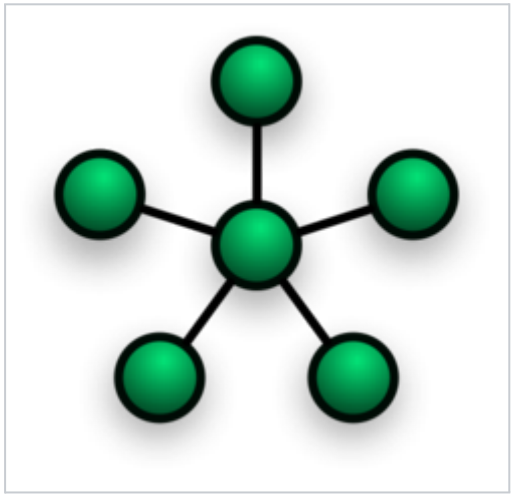
Topologia em ponto a ponto

A topologia ponto a ponto consiste na forma mais básica de interconexão de computadores, onde um par de computadores são interligados diretamente através de um meio de transmissão.

A topologia ponto a ponto pode ser empregada apenas para prover a conectividade entre dois dispositivos ou em redes de comunicação par a par. Em algumas situações, pode ser interessante interligar dois dispositivos diretamente para

trocar dados. Por exemplo, o sistema multimídia do seu carro poderia se conectar via Bluetooth com seu smartphone para permitir a execução de uma música em formato MP3 no sistema de som do carro. Além disso, essa topologia também pode ser usada para permitir a troca de dados entre pares de computadores não conectados à Internet por meio de um cabo crossover. Nessa configuração, são usados cabos de rede do tipo Cat5 ou Cat6 e a ligação dos pares internos do cabo par trançado são invertidos. Tanto no caso da conexão Bluetooth, quanto nas ligações com cabos crossover, a topologia ponto a ponto é empregada de forma física.[9]

Topologia em estrela



Topologia de rede em estrela

Neste tipo de rede, todos os usuários comunicam-se com um nodo (nó) central, que tem o controle supervisor do sistema, chamado *host*. Por meio do *host* os usuários podem se comunicar entre si e com processadores remotos ou terminais. No segundo caso, o *host* funciona como um comutador de mensagens para passar dados entre eles.

O arranjo em estrela é a melhor escolha se o padrão de comunicação da rede for de um conjunto de estações secundárias que se comunicam com o nó central. As situações nas quais isso acontece são aquelas em que o nó central está restrito às funções de gerente das comunicações e a operações de diagnósticos.

O gerenciamento das comunicações por este nó central pode ser por chaveamento de pacotes ou de circuitos.

O nó central pode realizar outras funções além das de chaveamento e processamento normal. Por exemplo, pode compatibilizar a velocidade de comunicação entre o transmissor e o receptor. Se o protocolo dos dispositivos fonte e destino for diferente, o nó central pode atuar como um roteador, permitindo duas redes de fabricantes diferentes se comunicar.

No caso de ocorrer falha em uma estação ou na ligação com o nó central, apenas esta estação fica fora de operação.

Entretanto, se uma falha ocorrer no nó central, todo sistema pode ficar fora do ar. A solução deste problema seria a redundância, mas isto acarreta um aumento considerável de custos.

A expansão de uma rede desse tipo só pode ser feita até um certo limite, imposto pelo nó central: em termos de capacidade de chaveamento, número de circuitos concorrentes que podem ser gerenciados e números de nós que podem ser servidos.

O desempenho obtido numa rede em estrela depende da quantidade de tempo requerido pelo nó central para processar e encaminhar mensagens, e da carga de tráfego de conexão, ou seja, é limitado pela capacidade de processamento do nó central.

Esta configuração facilita o controle da rede e a maioria dos sistemas de computação com funções de comunicação possuem um software que implementa esta configuração.

A topologia em estrela consiste em uma das topologias mais utilizadas atualmente. A maioria das redes LANs, seja de um escritório, uma casa ou uma universidade, possuem um conjunto de computadores que estão diretamente conectados a um roteador, switch ou hub para trocar dados. A diferença entre um roteador, switch e um hub consiste nas suas funcionalidades. Usando um switch, o pacote é recebido por uma porta e é encaminhado apenas para a porta em que se encontra o computador de destino. Enquanto que a função de um roteador consiste em processar o pacote e determinar sua rota, sempre que necessário.

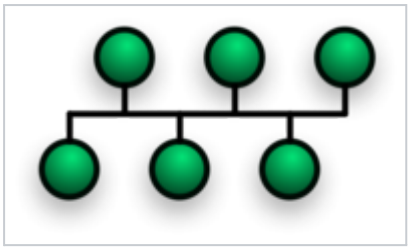
Além disso, muitos roteadores oferecem funcionalidades complementares, tais como o gerenciamento dos dispositivos das redes e da criação de redes virtuais. Apesar dessas diferenças, os hubs, switches e roteadores operam de maneira similar em uma topologia em estrela, interligando os dispositivos da rede e encaminhando o tráfego de uma origem para um destino. No caso das LANs, normalmente os roteadores, switches ou hubs também são conectados com um provedor de serviços de Internet, permitindo que os computadores da LAN possam acessar serviços disponibilizados na Internet.

Existem vantagens e desvantagens de empregar a topologia estrela. As principais vantagens consistem na facilidade de adicionar novos computadores, centralização do gerenciamento e a falha de um computador das bordas não afeta os demais computadores na rede. Como todos os computadores estão conectados no ponto central, a adição de um novo computador demanda apenas uma nova conexão ponto a ponto entre o novo computador e o dispositivo central.

Como as tarefas de encaminhamento de pacotes são delegadas ao componente central, podemos dizer que estas tarefas estão centralizadas neste dispositivo e sempre houver a necessidade de executar algum procedimento de alteração, adição ou exclusão de regras de encaminhamento, basta apenas acessar o dispositivo central, não necessitando modificações em outros componentes da rede. Além disso, como todas os computadores posicionados nas extremidades da topologia apenas enviam e recebem os pacotes, caso ocorra uma falha nestes dispositivos, a rede continuará a operar sem problemas.

Todavia, a desvantagem de organizar a rede por meio desta topologia consiste na possibilidade de falha do ponto central.

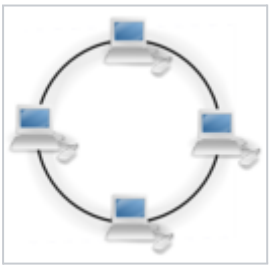
Topologia em barramento ou BUS



Topologia de rede em barramento - Simples

Topologia em barra comum é bastante semelhante ao conceito de arquitetura de barra em um sistema de computador, onde todas as estações (nós) se ligam ao mesmo meio de transmissão. Ao contrário das outras topologias, que são configurações ponto a ponto (isto é, cada enlace físico de transmissão conecta apenas dois dispositivos), a topologia em barra tem uma configuração multiponto.

Topologia em anel



Topologia de rede em anel

Na topologia anel como o próprio nome diz tem um formato circular e os computadores são conectados em série, formando um circuito fechado em forma de anel. Esse tipo de topologia não interliga os computadores diretamente porque existe uma série de repetidores conectados por um meio físico, onde cada estação é ligada a eles.

Na topologia em anel a organização da rede se dá de forma parecida com a topologia em estrela. Em vez de ter um concentrador no centro da rede em anel, há um dispositivo chamado Multistation Acess Unit ou MAU. A MAU tem a mesma função de um hub, mas trabalha com redes Token Ring em vez das

redes Ethernet e controla as comunicações entre os computadores de uma maneira ligeiramente diferente.[10]

- Vantagens

Um dos grandes benefícios da topologia anel é que ela é bem eficiente na transmissão de dados sem erros. Isso acontece porque apenas uma estação da rede consegue enviar dados por vez, o que diminui a chance de ocorrer uma colisão entre pacotes.

Em grandes redes, a topologia anel pode utilizar repetidores de sinal, aumentando a confiabilidade da transmissão e evitando a perda de dados. Além disso, o desempenho da rede não é prejudicado pelo aumento do volume de pessoas usuárias.[11]

- Desvantagens

Apesar de suas vantagens, a disposição em círculo apresenta uma grande vulnerabilidade: a falha de um dispositivo pode prejudicar a estabilidade de toda a rede. Nesse sentido, mesmo que você monitore a condição dos nodes, uma falha ainda pode acontecer em um deles, derrubando a conexão.

Além disso, a topologia anel não é tão recomendada para operações em crescimento. Afinal de contas, todos os dispositivos estão conectados e consumindo uma mesma banda. Sendo assim, a cada dispositivo adicionado, a rede aumenta o seu delay, justamente pelo maior número de estações pelo quais os dados precisarão passar.

Interface de rede

Um controlador de interface de rede (NIC) é um hardware de computador que fornece ao computador a capacidade de acessar a mídia de transmissão e tem a capacidade de processar informações de rede de baixo nível. Por exemplo, a NIC pode ter um conector para aceitar um cabo, ou uma antena para transmissão e recepção sem fio, e os circuitos associados.

O NIC responde ao tráfego dirigido a um endereço de rede para a NIC ou o computador como um todo.

Em redes Ethernet, cada controlador de interface de rede possui um único endereço de Controle de Acesso de Mídia (MAC) - geralmente armazenado na memória permanente do controlador. Para evitar conflitos de endereço entre dispositivos de rede, o Instituto de Engenheiros Elétricos e Eletrônicos (IEEE) mantém e administra a unicidade de endereço MAC. O tamanho de um endereço MAC Ethernet é de seis octetos. Os três octetos mais importantes são reservados para identificar os fabricantes NIC. Esses fabricantes, usando apenas seus prefixos atribuídos, atribuem de forma exclusiva os três octetos menos significativos de cada interface Ethernet que eles produzem.

Repetidores e hubs

Um repetidor é um dispositivo eletrônico que recebe um sinal de rede, o limpa de ruído desnecessário e o regenera. O sinal é retransmitido a um nível de potência mais alto, ou ao outro lado de uma obstrução, de modo que o sinal pode cobrir distâncias mais longas sem degradação. Na maioria das configurações de Ethernet de par trançado, são necessários repetidores para cabo que funciona com mais de 100 metros. Com as fibras ópticas, os repetidores podem estar a dezenas ou mesmo a centenas de quilômetros de distância.

Um repetidor com várias portas é conhecido como um hub Ethernet. Os repetidores trabalham na camada física do modelo OSI. Os repetidores requerem uma pequena quantidade de tempo para regenerar o sinal. Isso pode causar um atraso de propagação que afeta o desempenho da rede e pode afetar a função adequada. Como resultado, muitas arquiteturas de rede limitam o número de repetidores que podem ser usados em uma linha, por exemplo, a regra Ethernet 5-4-3.

Os hubs e repetidores nas LANs foram obsoletos principalmente por switches modernos.

Switches

Um switch de rede é um dispositivo que encaminha e filtra os datagramas da camada 2 OSI entre as portas com base no endereço MAC de destino em cada quadro. [16] Uma opção é distinta de um hub na medida em que apenas encaminha os quadros para as portas físicas envolvidas na comunicação em vez de todas as portas conectadas. Pode ser pensado como uma ponte multi-porto. [17] Aprende a associar portas físicas a endereços MAC examinando os endereços de origem dos quadros recebidos. Se um destino desconhecido for segmentado, o switch transmite para todas as portas, mas a fonte. Os switches normalmente possuem inúmeras portas, facilitando uma topologia em estrela para dispositivos e comutadores adicionais em cascata.

Os switches de várias camadas são capazes de rotear com base no endereçamento da camada 3 ou níveis lógicos adicionais. O termo switch é freqüentemente usado vagamente para incluir dispositivos como roteadores e pontes, bem como dispositivos que podem distribuir tráfego com base na carga ou com base no conteúdo da aplicação (por exemplo, um identificador de URL da Web).

Roteadores

Um roteador é um dispositivo de interconexão que encaminha pacotes entre redes processando as informações de roteamento incluídas no pacote ou datagrama (informações de protocolo da Internet a partir da camada 3). As informações de roteamento geralmente são processadas em conjunto com a tabela de roteamento (ou tabela de encaminhamento). Um roteador usa sua tabela de roteamento para determinar onde encaminhar pacotes. Um destino em uma tabela de roteamento pode incluir uma interface "nula", também conhecida como a interface do "buraco negro", porque os dados podem entrar nela, no entanto, nenhum processamento adicional é feito para os ditos dados, isto é, os pacotes são descartados.

Meio físico

O meio mais utilizado hoje é o Ethernet. O padrão Ethernet vem subdividido em: Coax/10BASE2, UTP (*Unshielded Twisted Pair* - Par Trançado Não Blindado)/10BASE-T e UTP/100baseT e Gigabit ethernet.

Também pode ser conectado por fibra óptica, um fino filamento contínuo de vidro com uma cobertura de proteção que pode ser usada para conectar longas distâncias.

E ainda há as redes sem fios, que se subdividem em diversas tecnologias: Wi-fi, bluetooth, wimax e outras.

Protocolos de comunicação

Um protocolo de comunicação é um conjunto de regras para trocar informações através de uma rede. Em uma pilha de protocolos (veja também o modelo OSI), cada protocolo aproveita os serviços do protocolo abaixo. Um exemplo importante de uma pilha de protocolos é HTTP (o protocolo da World Wide Web) executando TCP sobre IP (os protocolos da Internet) em relação ao IEEE 802.11 (o protocolo Wi-Fi). Esta pilha é usada entre o roteador sem fio e o computador pessoal do usuário doméstico quando o usuário está navegando na web.

Embora o uso de camadas de protocolo seja hoje omnipresente no campo da rede de computadores, tem sido historicamente criticado por muitos pesquisadores [20] por dois motivos principais. Em primeiro lugar, o resumo da pilha de protocolos dessa maneira pode causar uma camada superior para duplicar a funcionalidade de uma camada inferior, sendo um exemplo excelente a recuperação de erro tanto por base de link quanto de fim a extremo. [21] Em segundo lugar, é comum que uma implementação de protocolo em uma camada possa exigir dados, informações de estado ou de endereçamento que estejam apenas presentes em outra camada, derrotando o ponto de separação das camadas em primeiro lugar. Por exemplo, o TCP usa o campo ECN no cabeçalho IPv4 como indicação de congestion; IP é um protocolo de camada de rede, enquanto o TCP é um protocolo de camada de transporte.

Os protocolos de comunicação possuem várias características. Eles podem estar orientados para conexão ou sem conexão, eles podem usar o modo de circuito ou a troca de pacotes, e eles podem usar o endereçamento hierárquico ou o endereçamento plano.

Existem muitos protocolos de comunicação, alguns dos quais estão descritos abaixo.

IEEE 802

O é uma família de padrões IEEE que trata de redes de área local e redes de área metropolitana. O conjunto completo de protocolos IEEE 802 oferece um conjunto diversificado de recursos de rede. Os protocolos têm um esquema de endereçamento plano. Eles operam principalmente nos níveis 1 e 2 do modelo OSI.

Por exemplo, a ponte MAC (IEEE 802.1D) lida com o roteamento de pacotes Ethernet usando um protocolo Spanning Tree. O IEEE 802.1Q descreve VLANs e o IEEE 802.1X define um protocolo de controle de acesso à rede baseado em porta, que constitui a base para os mecanismos de autenticação usados nas VLANs (mas também é encontrado em WLANs) - é o que o usuário doméstico vê quando o o usuário deve inserir uma "chave de acesso sem fio".

Ethernet

A Ethernet, às vezes simplesmente chamada de LAN, é uma família de protocolos usados em redes LAN com fio, descritas por um conjunto de padrões, denominado IEEE 802.3, publicado pelo Instituto de Engenheiros Elétricos e Eletrônicos.

Wireless LAN

A LAN sem fio, também conhecida como WLAN ou WiFi, é provavelmente o membro mais conhecido da família de protocolos para usuários domésticos hoje. É padronizado pelo IEEE 802.11 e compartilha muitas propriedades com Ethernet com fio.

Internet Protocol Suite

O Internet Protocol Suite, também chamado TCP / IP, é a base de todas as redes modernas. Oferece serviços de ligação e serviços orientados para conexão em uma rede intrinsecamente não confiável atravessada por transmissão de grama de dados no nível de protocolo de Internet (IP). No seu núcleo, o conjunto de protocolos define as especificações de endereçamento, identificação e roteamento para o Protocolo de Internet Versão 4 (IPv4) e para IPv6, que, originalmente oficializada em 6 de junho de 2012, é a versão mais atual do Protocolo de Internet, que tendo uma capacidade de endereçamento muito mais ampla, veio com o objetivo a longo prazo de substituir o IPv4.

SONET/SDH

A rede óptica síncrona (SONET) e a Hierarquia Digital Síncrona (SDH) são protocolos de multiplexação padronizados que transferem múltiplos fluxos de bits digitais em fibra óptica usando lasers. Eles foram originalmente projetados para transportar comunicações de modo de circuito de uma variedade de fontes diferentes, principalmente para suportar codificação de voz em tempo real, descompactada e comutada em circuito no formato PCM (Modulação de Código de Pulso). No entanto, devido à sua neutralidade de protocolo e recursos orientados para o transporte, a SONET / SDH também foi a escolha óbvia para o transporte de quadros de Modo de Transferência Assíncrona (ATM).

Asynchronous Transfer Mode

Modo de transferência assíncrona (ATM) é uma técnica de comutação para redes de telecomunicações. Ele usa multiplexação assíncrona de divisão de tempo e codifica dados em pequenas células de tamanho fixo. Isso difere de outros protocolos, como o Internet Protocol Suite ou Ethernet que usam pacotes de tamanho variável ou quadros. O ATM tem similaridade com o circuito e a rede comutada por pacotes. Isso faz com que seja uma boa opção para uma rede que deve lidar tanto com o tráfego de dados de alto débito tradicional quanto com o conteúdo em tempo real e de baixa latência, como

voz e vídeo. ATM usa um modelo orientado a conexão em que um circuito virtual deve ser estabelecido entre dois pontos finais antes do início da troca de dados real.

Embora o papel do ATM esteja diminuindo em favor das redes da próxima geração, ele ainda desempenha um papel na última milha, que é a conexão entre um provedor de serviços de internet e o usuário doméstico.

Cellular standards

Existem vários padrões de celulares digitais diferentes, incluindo: Sistema Global para Comunicações Móveis (GSM), Serviço geral de rádio por pacotes (GPRS), cdmaOne, CDMA2000, Evolution-Data Optimized (EV-DO), taxas de dados aprimoradas para GSM Evolution (EDGE), Universal Mobile Telecommunications System (UMTS), Digital Enhanced Cordless Telecommunications (DECT), Digital AMPS (IS-136 / TDMA) e Integrated Digital Enhanced Network (iDEN).

Comparativo entre os modelos OSI e TCP/IP

Os dois modelos possuem diversas semelhanças:

- Baseados no conceito de pilha de protocolos independentes, ou seja, cada camada executa uma funcionalidade diferente;
- Ambos contêm camadas de aplicação;
- Ambos têm camadas de transporte e de rede com especificações comparáveis;
- As camadas dos modelos OSI e TCP/IP basicamente tem as mesmas funções.

Mas também tem várias diferenças:

- No modelo TCP/IP, os aspectos das camadas de sessão e apresentação estão inclusos na camada de aplicação;
- No TCP/IP, a camada física e de enlace do OSI são reunidas em uma única camada;
- O TCP/IP contém 4 camadas contra 7 camadas no modelo OSI.[4]

Escala geográfica

Uma rede pode ser caracterizada pela sua capacidade física ou pelo seu propósito organizacional. O uso da rede, incluindo a autorização do usuário e os direitos de acesso, diferem em conformidade.

Rede de nanoescala

Uma rede de comunicação em nanoescala possui componentes-chave implementados a nanoescala, incluindo portadores de mensagens, e alavanca princípios físicos que diferem dos mecanismos de comunicação macro escala. A comunicação em nanoescala estende a comunicação a sensores e atuadores muito pequenos, como os encontrados em sistemas biológicos e também tende a operar em ambientes que seriam muito difíceis para a comunicação clássica

Rede de área pessoal.

Uma Rede de área pessoal (PAN) é uma rede de computadores usada para comunicação entre computador e diferentes dispositivos tecnológicos de informação perto de uma pessoa. Alguns exemplos de dispositivos que são usados em um PAN são computadores pessoais, impressoras, aparelhos de fax, telefones, PDAs, scanners e até mesmo consoles de videogames. Um PAN pode incluir dispositivos com fio e sem fio. O alcance de um PAN normalmente se estende a 10 metros.[12] Um PAN com fio geralmente é construído com conexões USB e FireWire enquanto tecnologias como Bluetooth e comunicação por infravermelho tipicamente formam um PAN sem fio.

Rede residencial

Uma rede de área residencial (HAN) é uma LAN residencial usada para comunicação entre dispositivos digitais tipicamente implantados em casa, geralmente um pequeno número de computadores e acessórios pessoais, como impressoras e dispositivos de computação móvel. Uma função importante é o compartilhamento de acesso à Internet, muitas vezes um serviço

de banda larga através de um provedor de TV a cabo ou linha assinadora digital (DSL).

Rede de armazenamento

Uma rede de área de armazenamento (SAN) é uma rede dedicada que fornece acesso a um armazenamento consolidado de dados de nível de bloco. As SANs são usadas principalmente para criar dispositivos de armazenamento, como matrizes de disco, bibliotecas de fitas e jukeboxes ópticos, acessíveis aos servidores para que os dispositivos aparecem como dispositivos conectados localmente ao sistema operacional. Normalmente, uma SAN possui sua própria rede de dispositivos de armazenamento que geralmente não são acessíveis através da rede de área local por outros dispositivos. O custo e a complexidade das SANs caíram no início dos anos 2000 para níveis que permitiam maior adoção em ambientes empresariais e pequenas e médias empresas.

Rede de campus

Uma rede de área do campus (CAN) é constituída por uma interconexão de LANs dentro de uma área geográfica limitada. O equipamento de rede (switches, roteadores) e mídia de transmissão (fibra óptica, planta de cobre, cabeamento Cat5, etc.) são quase inteiramente de propriedade do inquilino / proprietário do campus (uma empresa, universidade, governo, etc.). Por exemplo, é provável que uma rede de campus universitário ligue uma variedade de edifícios do campus para se conectar a faculdades ou departamentos acadêmicos, a biblioteca e residências de estudantes.

Rede de espinha dorsal (backbone)

Uma rede de espinha dorsal é parte de uma infraestrutura de rede informática que fornece um caminho para a troca de informações entre diferentes LANs ou sub-redes. Uma espinha dorsal pode unir redes diversas dentro do mesmo edifício, em diferentes edifícios, ou em uma ampla área.

Por exemplo, uma grande empresa pode implementar uma rede espinha dorsal para conectar departamentos que estão localizados em todo o mundo. O equipamento que une as redes departamentais constitui o espinha dorsal da rede. Ao projetar uma rede de espinha dorsal, o desempenho da rede e o congestionamento da rede são fatores críticos para levar em consideração. Normalmente, a capacidade da rede backbone é maior que a das redes individuais conectadas a ele.

Outro exemplo de uma rede espinha dorsal é o espinha dorsal da Internet, que é o conjunto de redes de área ampla (WANs) e roteadores principais que vinculam todas as redes conectadas à Internet.

Rede metropolitana

Uma rede de área metropolitana (MAN) é uma rede de computadores de grande porte que normalmente abrange uma cidade ou um campus grande, com alcance na escala dos 10 quilômetros.

Meios de transmissões não guiados

Todas as comunicações sem fio de baseiam do principio da transmissão de ondas eletromagnéticas emitidas e recebidas pelo ar, por meio de antenas. No entanto ao invés de utilizar o termo "ar", especialistas da área utilizam a expressão "espectro eletromagnético" como referência ao ambiente em geral onde as ondas podem se expandir.[13]

Na ilustração podemos observar diferentes frequências classificadas como ondas eletromagnéticas não-ionizante e ionizantes.[13] Mas para entender esses conceitos, precisamos entender que todas as ondas eletromagnéticas transportam energia e viajam na velocidade da luz. As ondas eletromagnéticas ionizantes são aquelas que possuem energia suficiente para remover elétrons do átomo ou quebrar ligações químicas. Em contrapartida as ondas eletromagnéticas não-ionizantes não possuem energia suficiente para remover elétrons ou quebrar ligações químicas. Dentre as ondas

eletromagnéticas não-ionizantes, encontramos desde as frequências extremamente baixas (ELF), geralmente utilizadas em torres de comunicação cabeada, até as ondas de infravermelho. Separando as ondas não-ionizantes das ondas ionizantes, encontra-se a luz visível, a luz percebida pelos seres humanos. Na sequência, seguem as ondas eletromagnéticas ionizantes, compreendendo desde a radiação ultravioleta até os raios gama. As faixas de rádio, micro-ondas, infravermelho e luz visível do espectro eletromagnético podem ser usadas na transmissão de informações através da modulação da amplitude, da frequência, ou da fase das ondas. A modulação consiste em uma técnica criada para modificar as características da onda portadora do sinal. Na modulação por amplitude, a onda portadora do sinal é modificada em termos de amplitude para portar o sinal. A amplitude consiste na medida de oscilação da onda que pode ser positiva ou negativa. A modulação por frequência aplica técnicas para variar a frequência da onda portadora do sinal. A frequência indica o número de ocorrência de oscilações do sinal da onda em um determinado intervalo de tempo, sendo medida em Hz. A modulação de fase emprega técnicas para variar a fase da onda portadora do sinal. A fase de uma onda expressa o ângulo de uma onda.[13]

Tipos e meios de transmissão

Transmissão via satélite

- A transmissão via satélite é conduzida por ondas de alta frequência, porém quando se fala de ionosfera, as coisas acontecem diferentes, isso acontece por mudanças da direção de uma onda que se propaga em um determinado meio ao passar obliquamente para outro meio no qual a velocidade de propagação é alterada; anáclase [A refração ocorre em diferentes tipos de onda, embora seja mais comumente associada à luz]. Para que tudo ocorra normalmente é usado um satélite em órbita que tem o papel de ser um repetidor de micro-ondas no céu. Antenas transmissoras e Antenas recpetoras são

utilizadas nesse processo, cada uma com suas respectivas funções.

- **Antenas transmissoras:** Responsáveis por enviar micro-ondas para os satélites por meio do processo chamado Uplink. (É um componente de rede que permite que dois dispositivos de rede se conectem entre si com um cabo de padrão "reto", em vez de usar um de par trançado).
- **Antenas receptoras:** Essas recebem as micro-ondas pelo processo downlink. (processo de comunicação por satélite no qual os dados são enviados de um satélite para um terminal ou dispositivo terrestre, daí a palavra "para baixo").

Vale salientar que um satélite é composto por transponders, como também pode pode ser adicionado um processamento adicional para manipular ou redirecionar os feixes de dados separadamente. Assim, faz-se que seja mais difícil amplificar um sinal que poderia ser uma interferência [9].

Cabo de fibra óptica

As **fibras óticas** são filamentos flexíveis fabricados em materiais transparentes como fibras de vidro ou plástico e que são utilizadas como meio de propagação da luz. As fibras ópticas são geralmente muito finas, com apenas alguns **micrômetros de espessura** (10^{-6} m), mas podem ter vários quilômetros de comprimento. Fibras ópticas têm diversas aplicações, sendo a transmissão de dados uma das mais comuns.[14]

As fibras ópticas são formadas por um núcleo transparente de alto índice de refração revestido por camadas plásticas transparentes cujos índices de refração são mais baixos que os do núcleo. O fenômeno físico que permite a utilização das fibras ópticas é a **reflexão interna total** da luz.Para que ocorra a sua reflexão interna total, a luz é emitida para o interior do núcleo da fibra óptica em um **ângulo mínimo de incidência**, chamado de ângulo limite (também chamado de ângulo crítico), medido em relação à interface entre o núcleo e seu

revestimento. Tal ângulo permite que a luz sofra sucessivas reflexões internas no interior da fibra óptica sem que ela escape de lá. Dessa forma, a luz pode ser propagada por longas distâncias, com perdas mínimas em sua **intensidade**, além de acompanhar o formato em que os cabos de fibra óptica estão dispostos.

Há dois tipos de denominação recorrentes às fibras ópticas, os quais possuem características e finalidades próprias. Um deles é a fibra óptica **monomodo**. Esta apresenta um único caminho possível de propagação e é a mais utilizada em transmissão a longas distâncias (devido a baixas perdas de informação). Já a fibra **multimodo** permite a propagação da luz em diversos modos e é a mais utilizada em redes locais (LAN), devido ao seu custo moderado.[*carece de fontes*]

Outro aspecto importante das fibras ópticas consiste em como elas são conectadas. Existem três principais modos de conexões. No primeiro caso, as fibras podem possuir conectores nas extremidades do cabo e são plugadas em soquetes de fibra. A segunda forma de conexão compreende a união mecânica da fibra. Nesse caso, as extremidades são conectadas por meio de uma luva especial e um alinhamento cuidadoso pode ser realizado para maximizar a passagem do sinal. O terceiro modo de conexão compreende a fusão das extremidades. Usando a fusão proporciona um resultado muito semelhante a uma fibra sem emendas, todavia, existe uma pequena atenuação no sinal transmitido, mas mesmo assim este tipo de conexão se destaca em relação aos demais.[15]

THREADS E CONCORRÊNCIA

Monotarefa versus Multitarefa

Os primeiros sistemas operacionais suportavam a execução de apenas uma tarefa por vez. Nesse modelo, o processador, a

memória e os periféricos ficavam dedicados a uma única tarefa. Tínhamos um fluxo bem linear, como pode ser visto nesse diagrama:

Apenas no término da execução de uma tarefa que outra poderia ser carregada na memória e então executada.

O problema desse modelo é que enquanto o processo realizava uma operação de I/O para, por exemplo, ler algum dado do disco, o processador ficava ocioso. Ademais, uma operação do processador é infinitamente mais rápida que qualquer uma de leitura ou escrita em periféricos.

Para se ter ideia, quando falamos de uma operação que a CPU executa, lidamos com nanosegundos, enquanto em uma operação de rede consideramos mile segundos.

Se você “pingar” o Google observará isso:
Copiar

```
$ ping google.com.br
PING google.com.br (216.58.202.3): 56 data bytes
64 bytes from 216.58.202.3: icmp_seq=0 ttl=52 time=20.845 ms
...
```

Conforme as aplicações foram evoluindo, começou a se tornar um problema. Um exemplo clássico é um editor de texto que precisa executar diversas tarefas simultaneamente como, por exemplo, formatar o texto selecionado e verificar a ortografia dele, duas tarefas (simplificando) sendo executadas sobre a mesma “massa” de dados que é o texto selecionado.

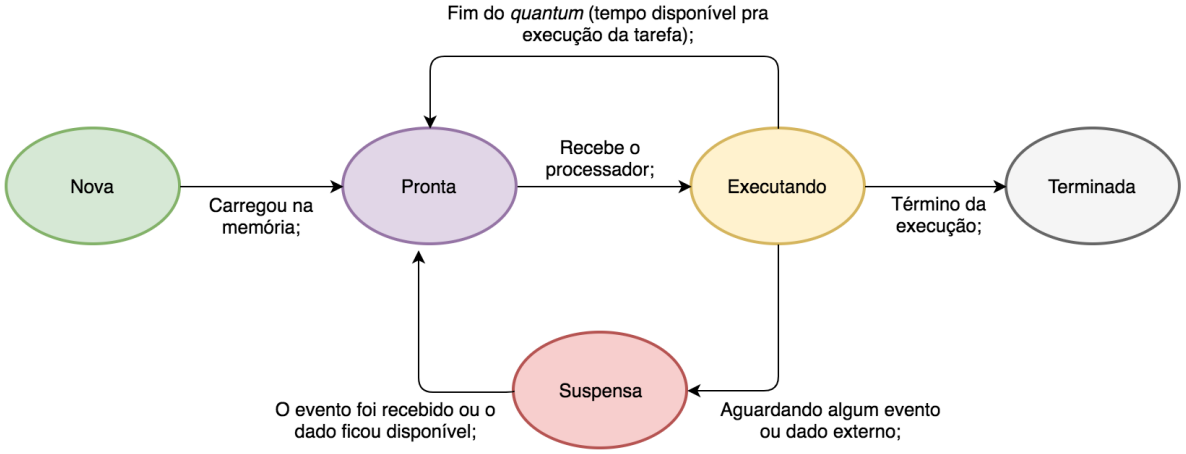
A solução encontrada para resolver esse problema foi permitir ao processador suspender a execução de uma tarefa que estivesse aguardando dados externos ou algum evento e passar a executar outra tarefa. Em outro momento de tempo, quando os dados estivessem disponíveis, a tarefa suspensa poderia ser retomada do ponto exato de onde ela havia parado. Nesse modelo, mais de um programa é carregado na memória. O mecanismo que permite a retirada de um recurso (o processador, por exemplo) de uma tarefa, é chamado de preempção.

Sistemas preemptivos são mais produtivos, ademais, várias tarefas podem estar em execução ao mesmo intervalo de tempo alternando entre si o uso dos recursos da forma mais justa que for possível. Nesse tipo de sistema as tarefas alteram de estado e contexto a todo instante.

Os estados de uma tarefa num sistema preemptivo:

- Nova: A tarefa está sendo criada (carregada na memória);
- Pronta: A tarefa está em memória aguardando a disponibilidade do processador para ser executada pela primeira vez ou voltar a ser executada (na hipótese de que ela foi substituída por outra tarefa, devido à preempção);
- Executando: O processador está executando a tarefa e alterando o seu estado;
- Suspensa: A tarefa não pode ser executada no momento por depender de dados externos ainda não disponíveis (dados solicitados à rede ou ao disco, por exemplo.);
- Terminada: A execução da tarefa foi finalizada e ela já pode sair da memória;

O diagrama de estado das tarefas com preempção de tempo:



Quantum pode ser entendido como o tempo que o sistema operacional dá para que os processos usem a CPU. Quando o quantum de um processo termina, mesmo que ele ainda não tenha terminado a execução de suas instruções, o contexto dele é trocado, é salvo na sua pilha de execução onde ele parou, os dados necessários e então ele volta pro estado de “pronto”, até que o sistema operacional através do seu escalonador de processos volte a “emprestar” a CPU pra ele (e então ele volta pro estado de “executando”). Trocas de contexto (de pronto pra executando etc) acontecem a todo momento. Milhares delas. É uma tarefa custosa para o sistema operacional, mas que é necessária para que a CPU não fique ociosa.

Um core (núcleo) do processador executa uma tarefa por vez, cabendo ao escalonador do sistema operacional cuidar dessa fila de tarefas, decidir quem tem prioridade, quem tem o quantum disponível etc.

O que é um processo?

Um processo pode ser visto como um container de recursos utilizados por uma ou mais tarefas. Processos são isolados entre si (inclusive, através de mecanismos de proteção a nível de hardware), não compartilham memória, possuem níveis de operação e quais chamadas de sistemas podem executar. Como os recursos são atribuídos aos processos, as tarefas fazem o uso deles a partir do processo. Dessa forma, uma tarefa de um

processo A não consegue acessar um recurso (a memória, por exemplo) de uma tarefa do processo B.
Um processo é uma entidade ativa, é a “instância” de um programa (entidade passiva). Podemos fazer analogia com orientação a objetos onde o programa seria a estrutura da classe e um processo seria um objeto instância dessa classe.

O kernel do sistema operacional possui descritores de processos, denominados PCBs (Process Control Blocks) e eles armazenam informações referentes aos processos ativos e cada processo possui um identificador único no sistema, conhecido como PID (Process IDentifier).

As tarefas de um processo podem trocar informações com facilidade, pois compartilham a mesma área de memória. No entanto, tarefas de processos distintos não conseguem essa comunicação facilmente, pois estão em áreas diferentes de memória. Esse problema é resolvido com chamadas de sistema do kernel que permitem a comunicação entre processos (IPC - Inter-Process Communication).

O que é uma thread?

Os processos podem ter uma série de threads associadas e as threads de um processo são conhecidas como threads de usuário, por executarem no modo-usuário e não no modo-kernel. Uma thread é uma “linha” de execução dentro de um processo. Cada thread tem o seu próprio estado de processador e a sua própria pilha, mas compartilha a memória atribuída ao processo com as outras threads “irmãs” (filhas do mesmo processo).

O núcleo (kernel) dos sistemas operacionais também implementa threads, mas essas são chamadas de threads de kernel (ou kernel-threads). Elas controlam atividades internas que o sistema operacional precisa executar/cuidar.

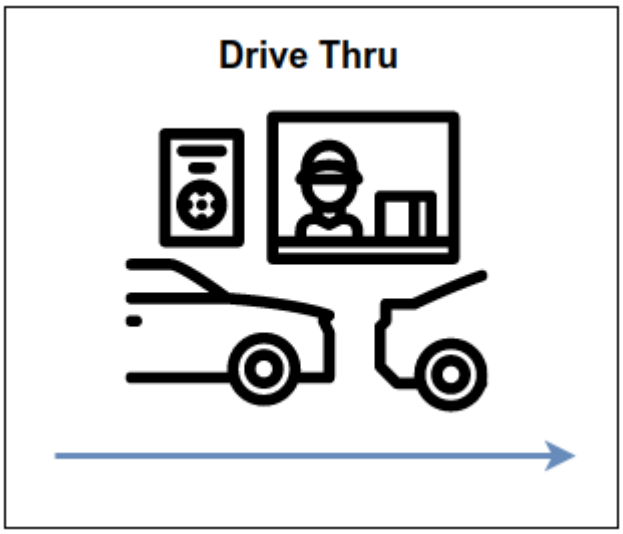
Concorrência e paralelismo

É comum achar que concorrência e paralelismo são a mesma coisa, mas não são. Rob Pike - um dos criadores da linguagem Go - em uma apresentação pontuou:
“Concurrency is about dealing with lots of things at once. Parallelism is about doing lots of things at once.”

Concorrência é sobre lidar com várias coisas ao mesmo tempo e paralelismo é sobre fazer várias coisas ao mesmo tempo. Concorrência é um conceito mais a nível de software e paralelismo mais a nível de hardware.

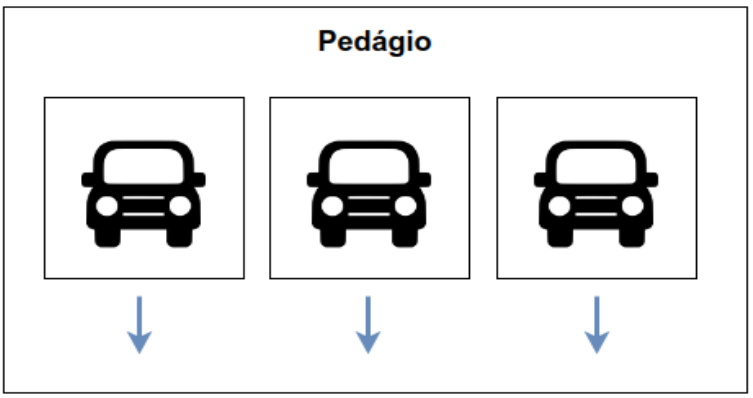
Concorrência é sobre a execução sequencial e disputada de um conjunto de tarefas independentes. Sob o ponto de vista de um sistema operacional, o responsável por esse gerenciamento é o escalonador de processos. Já sob o ponto de vista de concorrência em uma linguagem de programação como Go, por exemplo, o responsável é o scheduler interno da linguagem. Escalonadores preemptivos (como é o caso dos sistemas operacionais modernos) favorecem a concorrência pausando e resumindo tarefas (no caso de sistemas operacionais estamos falando de processos e threads no que chamamos de trocas de contexto) para que todas tenham a oportunidade de serem executadas.

Podemos fazer uma analogia em que concorrência (no contexto de um sistema operacional) é uma fila de drive thru em que carros estão disputando o recurso do atendimento, um por vez. Mas esse drive thru é especial, diferente do da vida real, ele é baseado num sistema preemptivo onde os carros possuem um tempo máximo em que podem ficar ali parados pelo atendimento, passando esse tempo, o carro tem que obrigatoriamente sair para dar oportunidade pro próximo carro da fila:



Paralelismo é sobre a execução paralela de tarefas, ou seja, mais de uma por vez (de forma simultânea), a depender da quantidade de núcleos (cores) do processador. Quanto mais núcleos, mais tarefas paralelas podem ser executadas. É uma forma de distribuir processamento em mais de um núcleo.

Paralelismo é aquele pedágio que permite que carros progridam em

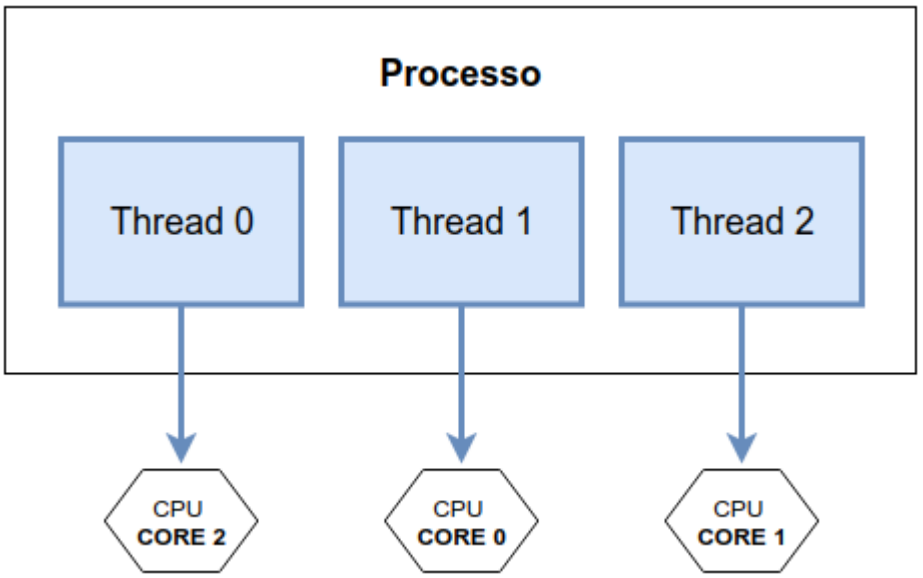


diferentes fluxos simultaneamente:

Também podemos dizer paralelismo é uma forma de atingir concorrência e que cada linha de execução paralela também é concorrente, pois os núcleos estarão sendo disputados por várias outras linhas de execução e quem gerencia o que o núcleo vai executar em dado momento do tempo é o escalonador de processos. Paralelismo implica concorrência, mas o contrário não é verdadeiro, pois é possível ter concorrência sem paralelismo, é só pensar no caso de uso de uma única

thread gerenciando milhares de tarefas, pausando e resumindo-as, esse é um modelo de concorrência sem paralelismo. Já o scheduler de corrotinas da linguagem Go é um exemplo de scheduler multi threaded, ele paraleliza a execução das corrotinas, ou seja, é um modelo de alta concorrência que faz uso dos núcleos do processador para paralelizar as execuções.

Do ponto de vista de um processador que possui mais de um núcleo (que é o padrão atualmente), um processo poderia ter, por exemplo, três threads rodando simultaneamente em três diferentes núcleos:



Síncrono e assíncrono

Síncrono e assíncrono são modelos de programação que estão intimamente ligados ao fluxo de execução, eles determinam como o código será escrito e como ele rodará. No modelo síncrono uma operação precisa ser finalizada para que outra tenha a oportunidade de ser executada. É um modelo linear, previsível, onde a execução acontece etapa por etapa e ele é base padrão da maior parte das linguagens de programação. Por exemplo, em PHP:

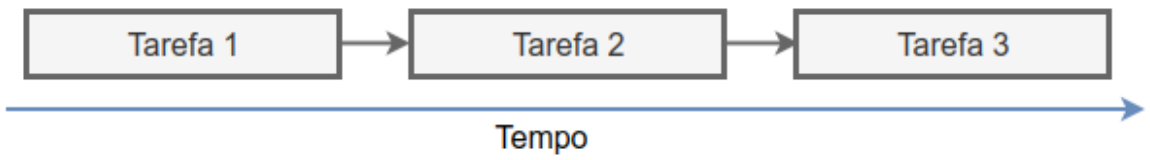
```
<?php
```

```php
function tarefa1() {
    echo "tarefa1\n";
}

function tarefa2() {
    echo "tarefa2\n";
}

tarefa2();

tarefa1();
```

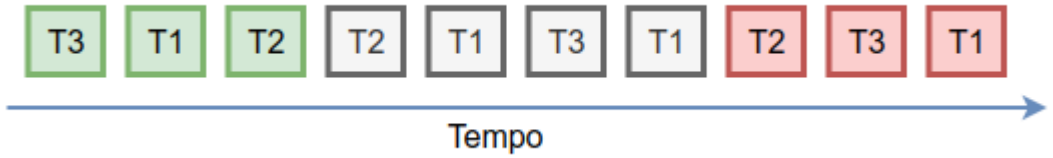
O resultado é previsível:
Copiar
tarefa2
tarefa1

Já no modelo assíncrono, uma operação não precisa esperar a outra ser finalizada, ao contrário disso, elas alternam o controle da execução entre si. É um modelo não previsível e que não garante a ordem da execução. É um modelo que favorece a concorrência.

Fazendo uma analogia, no modelo síncrono se você precisa colocar roupas para lavar e lavar louças, primeiro você poderia colocar as roupas para lavar e esperaria o tempo que fosse necessário até finalizar, para só depois lavar as louças. Enquanto que no modelo assíncrono você poderia colocar as roupas na máquina de lavar roupas e já começaria a lavar as louças no mesmo instante, pois você não precisa ficar ocioso esperando a máquina de lavar processar seu resultado para desempenhar outra tarefa, você pode pegar o "resultado" da máquina de lavar em um momento futuro.
Pensando no modelo síncrono de como as tarefas são executadas durante o tempo:



A tarefa 2 precisa obrigatoriamente esperar a tarefa 1 ser finalizada antes de ter a oportunidade de ser executada.
Já no modelo assíncrono nós temos a alternância da execução das tarefas, elas concorrem entre si:



Em verde as tarefas que iniciaram, em cinza a alternância de execução durante o tempo e em vermelho a finalização da execução delas. Fiz questão de não colocar em ordem pois assincronismo não garante ordem, assincronismo é sobre a execução de tarefas independentes.

Observe que tanto no modelo síncrono quanto no assíncrono uma tarefa é executada por vez, a diferença é que no modelo assíncrono a tarefa não precisa esperar o resultado da outra para poder ter seu tempo de execução. Assincronismo não necessariamente implica em paralelismo (se pensar em assincronismo num modelo multi-thread, sim, aí atinge paralelismo), assincronismo é uma forma de atingir concorrência.

Existem diferentes formas de se atingir assincronismo, sendo que a principal é a orientada a eventos (event loop com o padrão Reactor), modelo usado por NodeJS, Nginx, Swoole, ReactPHP entre outros. No caso do NodeJS e ReactPHP, por exemplo, eles fazem isso de forma single-thread, mas também é possível atingir assincronismo num modelo orientado a threads (multi-thread).

Em termos gerais, usar o modelo assíncrono é vantagem para a maior parte dos aplicativos que fazem uso intensivo de operações de I/O (leitura e escrita de arquivos, acesso a rede etc). Enquanto que o modelo de paralelizar processamento é mais indicado quando as tarefas são mais orientadas à CPU, quando elas precisam de mais poder de processamento/cálculo

que leitura e escrita em periféricos (na rede, ou sistema de arquivos etc).

Palavras finais

Esses assuntos são extensos e poderiam render bem mais conteúdo, mas isso deixaria a leitura engessada e complicada demais. Eu ainda acredito que a melhor forma de entendê-los realmente bem, é aplicá-los usando linguagens e frameworks. Por exemplo, no caso do PHP que é síncrono por padrão e que (ainda, na versão 7.4) não possui nenhum mecanismo a nível da linguagem para operações assíncronas, você pode estudar ReactPHP ou Swoole. A linguagem Go é uma boa pedida para entender na prática concorrência e paralelismo. E para estudar I/O assíncrono, NodeJS pode ser uma boa pedida.

ESCALANDO BANCO DE DADOS

DATABASE INDEXES

Os índices de banco de dados desempenham um papel fundamental na otimização do desempenho das consultas e na recuperação eficiente de dados de um banco de dados. Eles são estruturas de dados que aceleram a pesquisa de registros em uma tabela, funcionando de maneira semelhante a um índice em um livro, onde você pode encontrar rapidamente a página de referência de um tópico específico. Aqui está uma visão geral completa dos índices de banco de dados:

O que é um Índice de Banco de Dados:

- Um índice de banco de dados é uma estrutura de dados que fornece um meio eficiente de localizar registros em uma tabela de banco de dados. Ele contém valores

de coluna e um ponteiro para a posição física dos dados associados na tabela.

Benefícios dos Índices:

- Os índices aceleram as consultas, permitindo que o banco de dados localize registros mais rapidamente. Eles são especialmente úteis em tabelas grandes, onde a pesquisa sem um índice pode ser demorada.

Tipos de Índices:

- Diferentes bancos de dados suportam vários tipos de índices, incluindo:
  - Índices B-Tree: São usados para buscas exatas e faixas.
  - Índices Hash: São eficazes para igualdades exatas, mas não para intervalos.
  - Índices de Texto Completo: Usados para pesquisas de texto avançadas.
  - Índices Espaciais: Usados para dados geoespaciais.
  - Índices de BitMap: Usados para colunas com valores discretos.
  - Índices de Árvore Patrícia: Usados para pesquisas de prefixo.
  - Índices de Colunas Compostas: Combinam várias colunas em um único índice.

Colunas Indexadas:

- É importante escolher quais colunas indexar com base nas consultas frequentes e nos padrões de acesso aos dados. Índices em todas as colunas podem levar a sobrecarga e ocupação excessiva de espaço.

Criação de Índices:

- Os índices podem ser criados durante a criação da tabela ou posteriormente, usando declarações SQL específicas. A criação de índices adicionais pode ser necessária à medida que os requisitos de consulta evoluem.

Atualização e Manutenção de Índices:

- Os índices precisam ser atualizados quando os dados são inseridos, atualizados ou excluídos da tabela. Isso pode afetar o desempenho, uma vez que a manutenção de índices pode ser intensiva em recursos.

Custos e Overhead:

- Os índices não são gratuitos. Eles ocupam espaço em disco e podem afetar o desempenho durante operações de gravação. Portanto, é importante equilibrar os benefícios de desempenho com os custos associados aos índices.

Estatísticas de Índice:

- Os bancos de dados mantêm estatísticas sobre os índices para otimizar o planejamento de consultas. Isso inclui informações sobre a seletividade das colunas indexadas.

Uso de Índices em Consultas:

- Os otimizadores de consulta do banco de dados determinam se usarão índices ou uma varredura completa da tabela com base na complexidade da consulta, na seletividade do índice e em outros fatores.

Desempenho de Consulta:

- Índices melhoram o desempenho de consultas de igualdade, faixas e ordenação. No entanto, eles podem não ser eficazes em consultas que envolvem funções de manipulação de texto ou operações complexas.

Índices Clúster vs. Índices Não Clúster:

- Índices clúster determinam a ordem física dos registros na tabela. Cada tabela pode ter apenas um índice clúster. Índices não clúster são independentes da ordem física dos registros.

Monitoramento e Otimização:

- Os índices podem precisar ser ajustados ou reconstruídos periodicamente para manter o desempenho ideal do banco de dados.

Índices em Bancos de Dados Distribuídos:

- Em bancos de dados distribuídos, o uso de índices é mais complexo devido à distribuição dos dados em vários nós.

Índices Compostos (Composite Indexes):

- Índices compostos são criados em múltiplas colunas e podem ser úteis para consultas que envolvem várias colunas.

Fragmentação de Índice:

- Em bancos de dados grandes, a fragmentação de índice pode ocorrer, o que afeta o desempenho. A reorganização de índices pode ser necessária.

Os índices desempenham um papel crítico na otimização de consultas e na melhoria do desempenho do banco de dados. É importante entender como escolher, criar e manter índices de maneira eficaz para garantir que um aplicativo de backend seja capaz de recuperar dados de maneira eficiente, especialmente em cenários de grande volume de dados.

REPLICAÇÃO DE DADOS

Segundo Coulouris et al. (2013) replicação é a chave para prover alta disponibilidade e tolerância a falhas em sistemas distribuídos. A alta disponibilidade é um tópico de crescente interesse, principalmente com a atual tendência em direção à computação móvel e à operação desconectada. A tolerância a falhas é uma preocupação permanente dos serviços fornecidos em sistemas nos quais a segurança funcional (safety) é

fundamental e em outros tipos importantes de serviços e sistemas.
A replicação é uma técnica para melhorar serviços. As motivações para a replicação são: melhorar o desempenho de um serviço, aumentar sua disponibilidade ou torná-lo tolerante a falhas.

- Melhoria de desempenho: a colocação dos dados em cache de clientes e servidores é uma maneira conhecida de melhorar o desempenho.

A replicação de dados imutáveis é simples: ela aumenta o desempenho com pouco custo para o sistema. A replicação de dados mutáveis (dinâmicos), como os da Web, acarreta sobrecargas nos protocolos para garantir que os clientes recebam dados atualizados. Assim, existem limites para a eficácia da replicação como uma técnica de melhoria de desempenho.

- Tolerância a falhas: dados de alta disponibilidade não são necessariamente dados rigorosamente corretos. Eles podem estar desatualizados, por exemplo, ou dois usuários em lados opostos de uma rede que foi particionada podem fazer atualizações conflitantes e que precisem ser resolvidas. Em contraste, um serviço tolerante a falhas sempre garante comportamento rigorosamente correto, apesar de certo número e tipo de falhas. A correção está relacionada ao caráter atual dos dados fornecidos para o cliente e aos efeitos das operações do cliente sobre os dados. Às vezes, a correção também está relacionada ao tipo de respostas do serviço – como, por exemplo, no caso de um sistema de controle de tráfego aéreo, no qual dados corretos são necessários em escalas de tempo curtas.

A replicação de dados pode ser realizada via software, quando nos referimos há algumas linhas específicas numa determinada tabela ou ainda um banco de dados. Por outro lado, quando precisamos replicar um volume ou um disco, é comum recorrermos a replicação de hardware.
O uso da replicação permite o uso de vários níveis de granularidade. A replicação de banco de dados, por exemplo, permite desde a replicação de algumas linhas ou colunas

específicas até um conjunto de várias tabelas ou eventualmente um banco de dados inteiro.
A replicação de dados é uma das tecnologias mais empregadas para a construção de banco de dados distribuídos.

Funcionamento

A replicação é uma técnica para manter automaticamente a disponibilidade dos dados, a despeito das falhas do servidor. Se os dados são replicados em dois ou mais servidores que falham independentemente, então o software cliente pode acessar os dados em um servidor alternativo, caso o servidor padrão falhe ou se torne inacessível.

Na seguinte imagem, visualizamos uma implementação de replicação. Inicialmente a aplicação submete instruções de escrita (insert, update, delete) para o banco de dados.
Exemplo de replicação de dados

Inicialmente a aplicação submete instruções de escrita (insert, update, delete) para o banco de dados. Alterando, posteriormente esse banco de dados é replicado para suas quatro cópias, mantendo-as sincronizadas. Considerando que as réplicas são idênticas ao banco de dados original, uma consulta pode ser feita em qualquer uma das réplicas provendo assim cenários de alta disponibilidade e escalabilidade. A indisponibilidade do banco principal, não compromete as consultas. Uma vez que o banco principal esteja disponível, é possível redirecionar as operações de escritas para uma das réplicas.

Aplicabilidade

- Prover disponibilidade do dado em outros sites no caso da queda do principal.
- Diminuir a contenção em um único site, uma vez que o dado em múltiplos sites evita a concentração de acessos.
- Possibilitar processamento paralelo distribuído entre os sites.
- Melhorar o desempenho aproximado do dado do seu consumidor, evitando assim atrasos na transmissão em links considerados lentos.
- Permitir o uso dos dados em aplicações desconectadas, pois essas podem trabalhar de forma off-line e replicar os dados posteriormente.

Arquitetura de uma replicação

Servidores trabalham com um modelo de gerenciador de réplicas, esse gerenciador tem o papel de:
Coordenar o processo de replicação:

- Manter a consistência de estado e transparência do conjunto dos servidores.
- Manter o controle da concorrência;
- E fazer o controle de falhas de servidores.

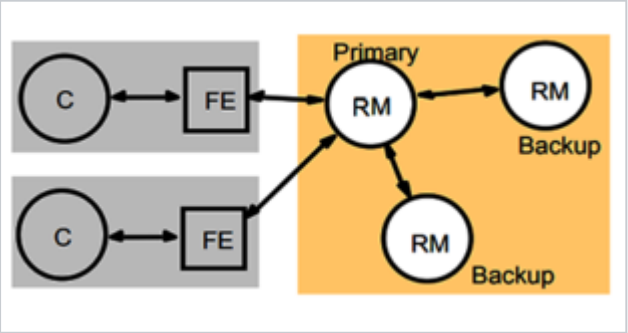
Tipos de Replicação:

- Replicação passiva;
- Replicação ativa.

Replicação Passiva (backup primário)

No modelo passivo, ou de backup primário, de replicação para tolerância a falhas, existe, em determinado momento, um único gerenciador de réplica (RM) primário e um ou mais gerenciadores de réplica secundários – backups ou escravos. Na forma pura do modelo, os front-ends (FE) se comunicam somente com o gerenciador de réplica primário para obterem o serviço. O gerenciador de réplica primário executa as operações e envia

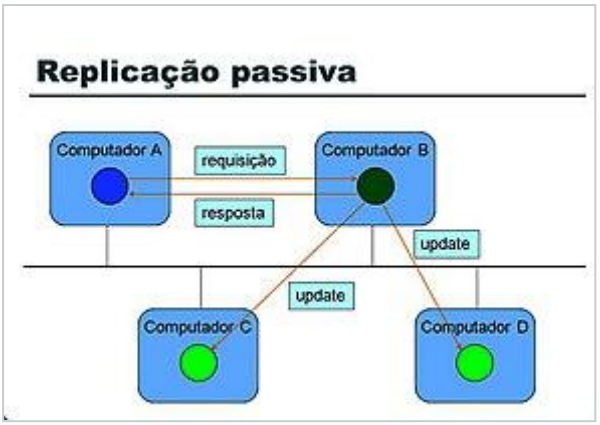
cópias dos dados atualizados para os backups. Se o primário falhar, um dos backups será promovido para atuar como primário.



O modelo passivo (backup primário) de tolerância a falhas.

1. A sequência de eventos, quando um cliente solicita que uma operação seja executada, é a seguinte:
2. Requisição: o front-end emite a requisição, contendo um identificador exclusivo, para o gerenciador de réplica primário.
3. Coordenação: o gerenciador primário trata cada requisição atomicamente, na ordem em que a recebe. Ele verifica o identificador exclusivo, para o caso de já a ter executado e, se assim for, simplesmente envia a resposta novamente.
4. Execução: o gerenciador primário executa a requisição e armazena a resposta.
5. Acordo: se a requisição é uma atualização, o gerenciador primário envia o estado atualizado, a resposta e o identificador exclusivo para todos os gerenciadores de backup. Os gerenciadores de backup enviam uma confirmação.
6. Resposta: o gerenciador primário responde para o front-end, o qual envia a resposta de volta para o cliente.

Detecção de falha do primário



Detecção de falha no backup primário

- Cliente estabelece timeout para requisição;
- Backups fazem verificação (keepalive).

Solução: Um novo primário é eleito.

Casos:

- Antes de iniciar o update;
- Durante ou após o update, mas antes de enviar a resposta ao cliente;
- Após enviar a resposta ao cliente.

Vantagens e desvantagens da Replicação Passiva

Vantagens:

- Réplicas não precisam obter sempre o mesmo efeito para uma certa requisição.
- Interação simples entre cliente e servidor.

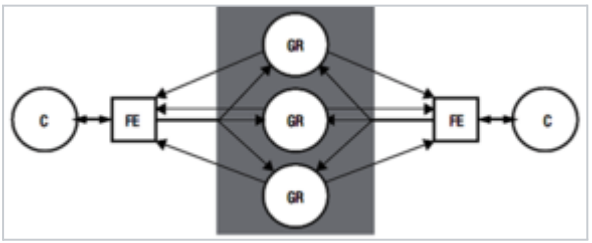
Desvantagens:

- A frequência de checkpoints pode prejudicar o desempenho do serviço replicado.
  Solução: fazer update a cada n requisições.
- Cliente retransmite requisições entre o último update realizado e a falha do primário.
- Um mecanismo de log em disco armazena todas as requisições desde o último checkpoint.

Replicação Ativa

No modelo ativo de replicação para tolerância a, os gerenciadores de réplica são máquinas de estado que desempenham papéis equivalentes e são organizados como um

grupo. Os front-ends enviam suas requisições por multicast para o grupo de gerenciadores de réplica, e todos os gerenciadores de réplica processam a requisição independentemente, mas de forma idêntica, e respondem. Se qualquer gerenciador de réplica falhar, isso não tem nenhum impacto sobre o desempenho do serviço, pois os gerenciadores de réplica restantes continuam a responder normalmente. Veremos que a replicação ativa pode tolerar falhas bizantinas, pois o front-end pode reunir e comparar as respostas que recebe.



Replicação ativa

Sob a replicação ativa, a sequência de eventos quando um cliente solicita a execução de uma operação é a seguinte:

1. Requisição: o front-end anexa um identificador exclusivo à requisição e a envia por multicast para o grupo de gerenciadores de réplica, usando uma primitiva de multicast totalmente ordenada e confiável. Supõe-se que o front-end pode falhar por colapso, na pior das hipóteses. Ele não emite a próxima requisição até que tenha recebido uma resposta.
2. Coordenação: o sistema de comunicação em grupo envia a requisição para cada gerenciador de réplica correto na mesma ordem (total).
3. Execução: todos os gerenciadores de réplica executam a requisição. Como eles são máquinas de estado, e como as requisições são distribuídas na mesma ordem total, todos os gerenciadores de réplica corretos processam a

requisição de forma idêntica. A resposta contém o identificador de requisição exclusiva do cliente.
4. Acordo: nenhuma fase de acordo é necessária, devido à semântica da distribuição por multicast.
5. Resposta: cada gerenciador de réplica envia sua resposta para o front-end. O número de respostas reunidas pelo front-end depende das suposições de falha e do algoritmo de multicast. Se, por exemplo, o objetivo for tolerar apenas falhas por colapso e o multicast satisfizer o acordo uniforme e as propriedades de ordenação, o front--end passará ao cliente a primeira resposta que chegar e descartará as restantes (ele pode distingui-las das respostas de outras requisições examinando o identificador presente na resposta).

Recuperação de falhas

A recuperação é feita através de uma:

- Atualização do estado;
- Execução das requisições perdidas.

Exige que as réplicas tenham comportamento determinístico: A requisição deve produzir o mesmo efeito em todas as réplicas. Caso contrário, pode ocorrer divergência de estados.

Exige atomicidade na comunicação: Uma certa requisição é recebida por todas as réplicas ou, então, não é recebida por nenhuma réplica.

Exige ordenação na comunicação: Todas as réplicas recebem as mensagens (requisições) na mesma ordem.

Vantagens e desvantagens da Replicação Ativa

Vantagens

- Falhas são mascaradas quase que instantaneamente.

- Adequada para aplicações que exigem serviços ininterruptos e com sobrecarga mínima em situações de falha, como aplicações de tempo real.
- Cobre um amplo espectro de faltas: crash, omissão, temporização e valor.

Desvantagens

- Tem alto custo na comunicação.
- Exige muitos recursos do sistema (memória, processador).

Principais desafios da replicação

Os dados em grandes corporações estão cada vez mais volumosos. O principal entrave é que o tamanho da largura da banda, necessária para trafegar as informações, não acompanha o crescimento na mesma proporção que o armazenamento. As estratégias de replicação ativo-ativo são, portanto, as mais interessantes, mesmo requerendo mudanças de softwares para o funcionamento transparente em data centers. Esses são desafios que novas tecnologias de banco de dados, como o NoSQL e ferramentas de big data, são capazes de resolver de maneira nativa, sem que haja a necessidade de aquisição de softwares específicos para essa finalidade.

Existem, ainda, alternativas que permitem que a replicação seja feita em ambientes internos com um único datacenter. Um bom exemplo disso são os de Hadoop, que replicam as informações, por padrão, três vezes dentro de seu próprio espaço. Para empresas menores, onde não há a possibilidade de investimento em outros datacenters, é importante garantir a alta disponibilidade por meio de serviços com arquitetura nativa de replicação, como a NoSQL e as ferramentas de big data. Hoje a melhor forma de se armazenar, recuperar e analisar as informações em um único ambiente utilizando big data é por meio de sistemas preparados de forma nativa para conseguir obter resultados de alta disponibilidade com a replicação de dados.

Atualidade

Atualmente, as grandes corporações não podem se dar ao luxo de terem suas operações interrompidas, independentemente do motivo. A rápida recuperação de sistemas após desastres naturais, como enchentes, incêndios, panes elétricas, terremotos, furacões ou tempestades é uma das maiores preocupações nas companhias. Soluções que permitem a continuidade operacional, mesmo após um evento catastrófico, são cada vez mais bem vistas e utilizadas.

Nesse aspecto, a replicação de dados torna-se uma saída interessante para as empresas que buscam a rápida recuperação depois de uma fatalidade. Essas ferramentas são usadas, normalmente, por companhias que possuem mais de uma estrutura de data center. Assim, se ocorrer um problema em um deles, é possível continuar com as operações normalmente a partir de outro centro de processamento. Para a implementação, o custo pode ser considerado alto.

Nesses casos, é importante que as empresas ponderem se o valor por ter a operação parada por dias ou semanas é menor do que o de implantação de replicação. Dependendo da conclusão, é fundamental que haja uma reavaliação das estratégias de recuperação, validando a necessidade de um novo datacenter. Esse, por sua vez, normalmente deve ser instalado em região geográfica distante do original.

ORMs

Object-Relational Mapping (ORM), em português, mapeamento objeto-relacional, é uma técnica para aproximar o paradigma de desenvolvimento de aplicações orientadas a objetos ao

paradigma do banco de dados relacional. O uso da técnica de mapeamento objeto-relacional é realizado através de um mapeador objeto-relacional que geralmente é a biblioteca ou framework que ajuda no mapeamento e uso do banco de dados.

Problema da impedância de dados

Quando estamos trabalhando com aplicações orientadas a objetos que utilizam banco de dados relacionais para armazenamento de informações, temos um problema chamado impedância objeto-relacional devido às diferenças entre os 2 paradigmas.

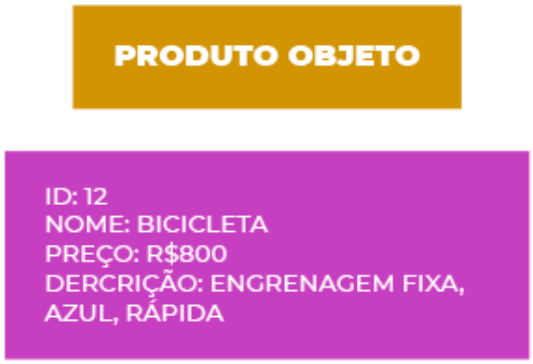
O banco de dados relacional trabalha com tabelas e relações entre elas para representar modelos da vida real. Dentro das tabelas temos várias colunas e a unidade que temos para representação no modelo relacional é uma linha:

TABELA: PRODUTO

ID	NOME	PREÇO	DESCRIÇÃO
12	BICICLETA	R$800	ENGRENAGEM FIXA, AZUL, RÁPIDA
13	CAPACETE	R$20,99	PRETO, AJUSTÁVEL
14	UNIFORME	R$35	PEQUENO (FEMININO), VERDE E BRANCO

O paradigma orientado a objetos possui um modo um pouco diferente de trabalhar. Nele nós temos diversos elementos como classes, propriedades, visibilidade, herança e interfaces. A

unidade quando falamos de orientação a objetos é o objeto que representa algo do mundo real, seja abstrato ou concreto:

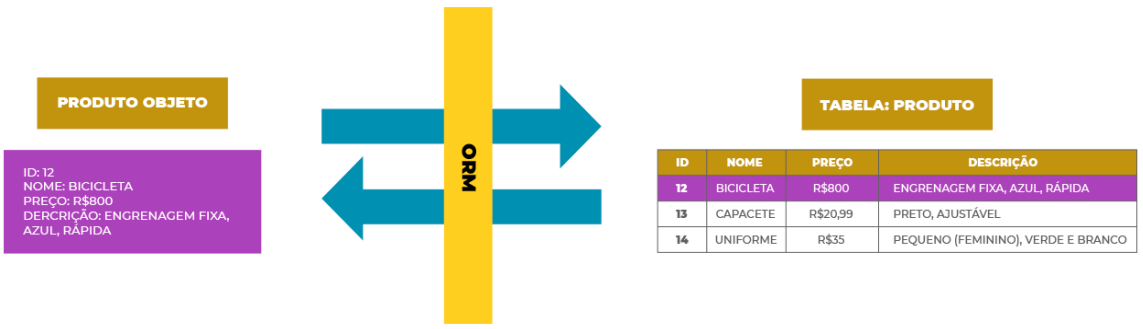


As principais dificuldades que essas diferenças entre paradigmas causa:

- Representação dos dados e do modelo, já que as estruturas são distintas;
- Mapeamento entre os tipos de dados da linguagem de programação e do banco de dados;
- Modelo de integridade relacional do banco relacional;

O ORM

Pensando nos problemas descritos acima, o ORM define uma técnica para realizar a conciliação entre os 2 modelos. Uma das partes centrais é através do mapeamento de linhas para objetos:



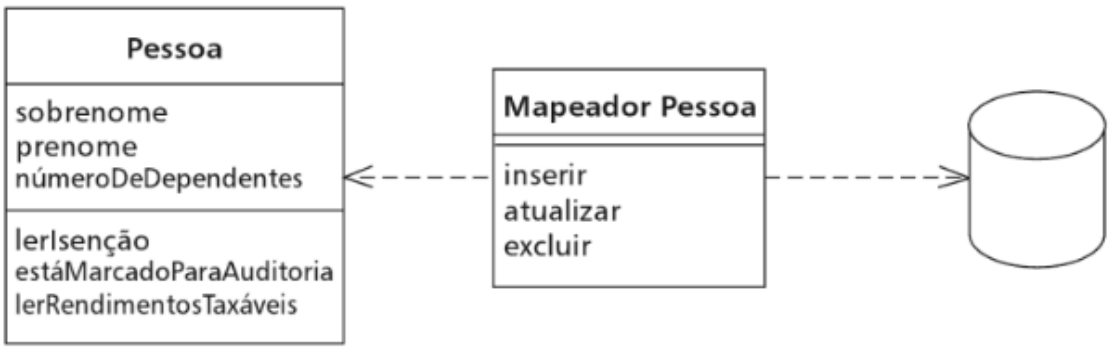
As bibliotecas ou frameworks ORM definem o modo como os dados serão mapeados entre os ambientes, como serão acessados e gravados. Isso diminui o tempo de desenvolvimento, uma vez que não é necessário desenvolver toda essa parte. Outra vantagem está na adaptação de novos membros na equipe, como muitos projetos comerciais utilizam a mesma ferramenta, é possível encontrar membros que já estão acostumados com o padrão de trabalho.

Padrões utilizados no mercado

Independente da linguagem de programação que o ORM é implementado, geralmente ele segue um padrão bem definido. No mercado existem dois padrões que são amplamente utilizados, o Data Mapper e o Active Record. Ambos os padrões foram definidos por Martin Fowler em seu livro Padrões de Arquitetura de Aplicações Corporativas.

Data Mapper

Nesse padrão a classe que representa a tabela do banco de dados não deve conhecer os recursos necessário para realizar as transações com banco de dados: inserir, atualizar e apagar informações. Esses recursos ficam em uma classe própria do ORM, garantindo que as classes que representam a tabela tenha uma única responsabilidade:



(Imagem do livro “Padrões de Arquitetura de Aplicações Corporativas”)

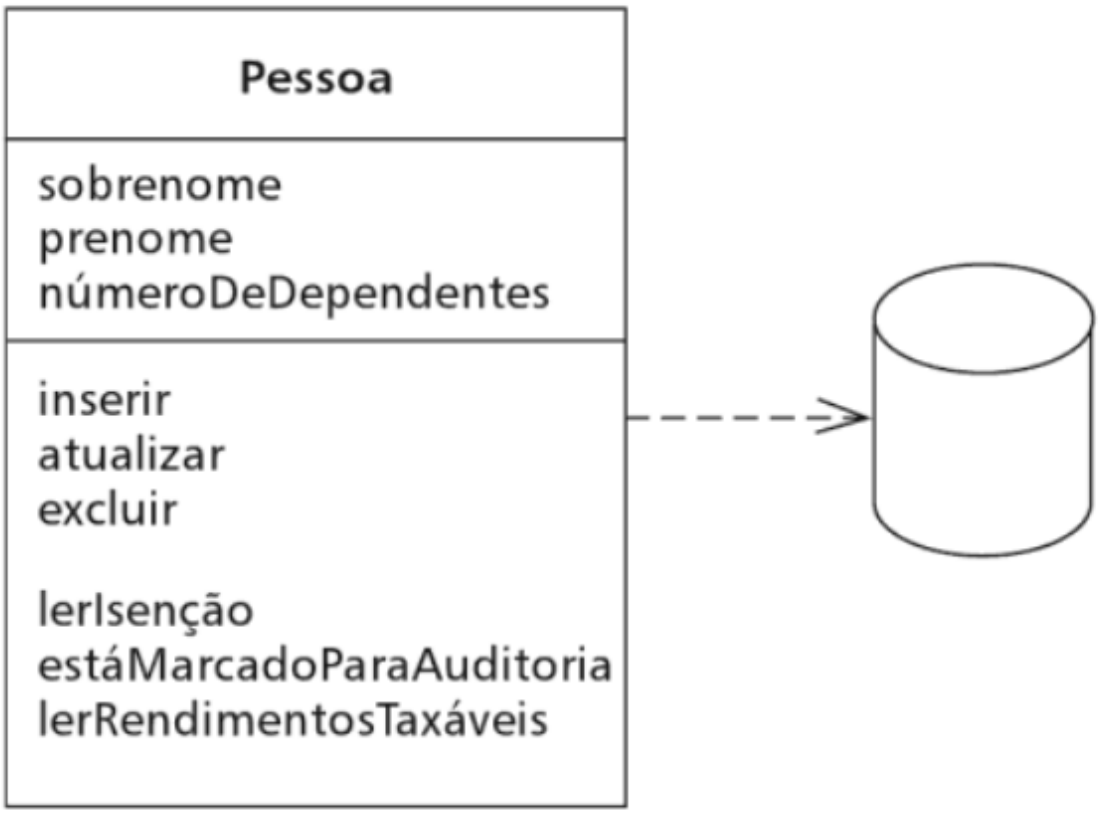
Na prática, para a maioria dos ORMs do mercado que implementam o padrão Data Mapper, independente da linguagem, vamos ter um código muito parecido com abaixo:

```
EntityManager entityManager =
Persistence.createEntityManagerFactory("persistente-unit");

entityManager.getTransaction().begin();

Pessoa pessoa = new Pessoa();
pessoa.setId(1);
pessoa.setSobrenome("Silva");
pessoa.setPrenome("João");
pessoa.setNumeroDeDependentes(2);

entityManager.persist(pessoa);

entityManager.getTransaction().commit();
```

O código acima é um exemplo do Hibernate para Java.

Active Record

Nesse padrão, diferentemente do anterior, a classe que representa a tabela conhece os recursos necessários para realizar as transações no banco de dados, geralmente ela herda uma classe com todos esses comportamentos. Veja abaixo um diagrama retirado do livro “Padrões de Arquitetura de Aplicações Corporativa”:



Na maioria dos ORM que implementam o padrão Active Record teremos um código muito parecido com esse:

```
pessoa = pessoa.new

pessoa.sobrenome = "Silva"

pessoa.prenome = "João"

pessoa.numeroDeDependentes = 2


pessoa.save()
```

O código é válido para o ORM do Ruby On Rails.

Principais ORMs do mercado

Java

- Hibernate
- EclipseLink

- ActiveJPA

Kotlin

- Exposed

C#

- Entity Framework
- Nhibernate
- Dapper

Node

- Sequelize

PHP

- Doctrine
- Eloquent

Ruby

- Ruby On Rails ActiveRecord
- Datamapper

Python

- DjangoORM
- SqlAlchemy

Quem é melhor? ORM ou SQL puro?

1. Introdução

Se você já aprendeu a construir uma aplicação, nem que seja simples, provavelmente já ouviu falar de banco de dados. Por mais que você possa não saber o que é isso, eu vou te explicar

de maneira superficial (coloca nos comentários se você quer um post especialmente sobre banco de dados).
"Banco de dados é um local é onde se guarda os dados de uma aplicação".
Alguns bancos de dados são SQL (Structure Query Language) outros NoSql, mas vamos focar aqui nesse post em SQL, ok?
Ok, sabendo disso, há situações no dia a dia de desenvolvimento que pode causar um impasse em algumas pessoas, como por exemplo: "Desenvolver o banco e efetuar operações com

SQL puro

Caso você não saiba o que é isso, basicamente é a linguagem de estilo funcional para bancos de dados relacionais, ou seja, com ela você informa para o banco o que fazer e não como fazer. Com ela conseguimos salvar os dados no banco, atualizar, deletar e efetuar até mesmo busca específicas sobre os dados.

SQL é extremamente importante para a vida de desenvolvedor, ora, se você nunca sofreu um pouquinho criando conexões e lidando com transações jdbc da vida, você ainda não viveu o suficiente kkkk, mas brincadeiras a parte, é muito interessante saber como utilizar ela pois ela é a base para entender ORM.

O que é ORM?

ORM é uma sigla para Object-Relational Mapping, ou seja, Mapeamento Objeto Relacional. ORM por si só é uma técnica de permite que as aplicações trabalhem com objetos em vez de lidar com tabelas sql por cima dos panos. Entretanto, geralmente quando falamos de ORM, estamos nos referindo a ferramentas de desenvolvimento, os famosos frameworks ORM.

Existem diversos frameworks ORM no mercado como o famoso Prisma, Elouquent, Hibernate etc. Ele é muito útil no dia a

dia, pois possibilita padronizar e automatizar tarefas rotineiramente repetitivas no cotidiano de dev.

Agora que você já sabe superficialmente o que são ambos, tenha fixado na cabeça uma coisa "Nem tudo são flores". Usar SQL puro tem suas vantagens, muito boas por sinal, mas também não é um mar de rosas. A mesma ideia serve para ORM, pois há muitas vantagens como produtividade de desenvolvimento, entretanto, há algumas desvantagens no seu uso a depender do caso.

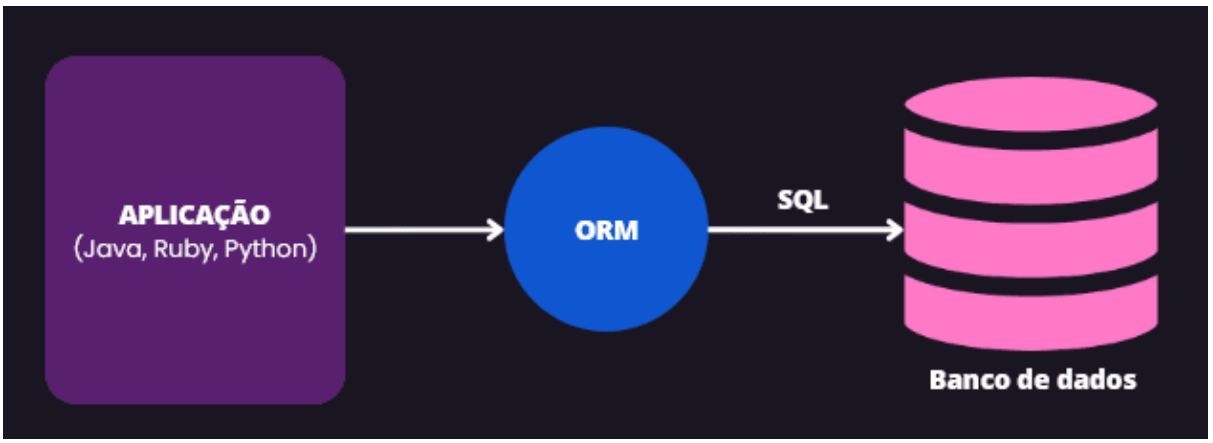
"Mas o que são essas vantagens e desvantagens na real? Dá pra fazer algum comparativo entre ambos?", neste post eu pretendo trazer 3 pontos que eu julguei interessantes para ser debatidos (caso discorde, comenta aí) são eles: Performance, produtividade e escalabilidade.

Performance

Sim, comecei por uma das partes mais polêmicas quando se trata de ORM. "Mas por quê?" Algumas pessoas afirmam que ORM é muito menos performático do que usar SQL puro. De fato, não há como negar que ter uma camada de abstração em uma infraestrutura como a camada de dados aumenta a latência, isso é inquestionável. Porém, entretanto e todavia, essa diferença de latência pode ser desprezível a depender do nível da tua aplicação.

"Calma, que raios de camada é essa que você falou?", segue o fio das imagens:

Na imagem acima você consegue ver como é o processo com SQL puro na sua aplicação. Resumindo, você terá um driver de conexão que permite a conexão direta com o banco de dados.



Já nesta imagem você consegue ver claramente que há uma camada a mais, camada essa que pode gerar um overhead adicional na aplicação, nossa camada de ORM, uma abstração a mais.

"Mas pera... em um caso hipotético que eu tenho uma aplicação de CRUD para uma padaria da minha vizinhança, qual é melhor? julgando unicamente Performance?", já que você está com teimosia, vamos julgar somente nesse critério. Bom... a resposta é que não faz diferença nesse caso kkkk. A latência só influencia mais em aplicações mais robustas que precisam mais do que um CRUD simples ou consultas simples.

"Ah Lucas, mas meu papel como desenvolvedor não é fazer o melhor software possível?", lembre-se que decisões não são

tomadas levando somente um fator em consideração, deve ser levado em consideração a produtividade da equipe (ou euquipe), tempo de entrega, requisitos do sistema etc. Criar software não é só pegar umas tecnologias aleatórias e implementar, vai muito além do que isso.

O famoso N+1

"Ei, eu vi uma coisa uma vez sobre o problema N+1 que os ORM's têm!", bem lembrado, esses frameworks de persistência geralmente possui um "problema" de consultar mais do que devem, não entendeu? então se liga nesse caso:
Você criou um CRUD relacional simples com o hibernate em que um cliente possui vários pedidos.
código das classes:

```
@Entity
public class Cliente {
    @Id
    private Long id;
    private String nome;

    @OneToMany
    private List<Pedido> pedidos;
    // getters e setters
}

@Entity
public class Pedido {
    @Id
    private Long id;
    private BigDecimal valor;

    @ManyToOne
    private Cliente cliente
    // getters e setters
}
```

Depois de ter criado as classes que serão as representações de tabelas no banco, vamos para o cenário em que você chama um findAll para clientes, ou seja, você busca uma lista de todos os clientes que estão contidos no banco de dados.
as queries geradas pelo ORM seriam:

```
SELECT * FROM CLIENTE;

SELECT * FROM PEDIDO WHERE ID_CLIENTE = 1;
SELECT * FROM PEDIDO WHERE ID_CLIENTE = 2;
SELECT * FROM PEDIDO WHERE ID_CLIENTE = 3;
SELECT * FROM PEDIDO WHERE ID_CLIENTE = 4;
```

O ORM entende que primeiro deve fazer a query SELECT * FROM CLIENTE, mas daí, depois que ele fez, ele percebe que precisa agora dos pedidos do cliente, ou seja, para cada uma consulta de buscar todos os clientes, ele tem que ir mais N vezes no banco para pegar os dados agregados.

"Por quê o ORM trabalha desse jeito? tá só gerando query indesejada, seria muito mais vantajoso fazer um INNER JOIN com SQL puro", de fato tá criando queries indesejadas, deixando o sistema mais suscetível a gargalos futuros, mas há solução para isso, geralmente é simples a correção disso. A curva de aprendizado para sacar que esse problema existe e solucionar não é grande, principalmente se a pessoa souber manipular SQL.

ORM's geralmente possuem uma flexibilidade muito boa para solucionar esse tipo de problema, como por exemplo, escrevendo uma query na linguagem do framework ou escrevendo SQL puro dentro do ORM.

Produtividade

Nem só de performance vive o homem (na minha cabeça isso teve graça, mas enfim). Produtividade é um fator muito relevante quando se trata de tomada de decisão, sem ele, não conseguimos ter uma boa dimensão do que vale a pena ou não.

SQL pode ser muito produtivo em termos de escrita de queries quando há uma equipe que sabe muito bem o que está fazendo. No entanto, em termos de manutenção, pode ser um desafio, especialmente quando há uma leve alteração na estrutura do banco de dados ou uma migração para um SGBD (Sistema Gerenciador de Banco de dados) diferente. Desenvolvedores podem usar recursos específicos de um determinado SGBD, como PostgreSQL, Oracle ou MySQL, para alcançar um desempenho melhor, mas isso pode se tornar uma dor de cabeça na hora de migrar para outro SGBD. Além disso, uma mudança em uma coluna no banco de dados pode exigir alterações em várias queries para se adequar à mudança.

"Mas e o ORM?" - Bem, há uma vantagem muito grande em termos de produtividade que os ORM's trazem para nós, principalmente os mais conhecidos. Os ORM's permitem uma facilidade em termos de padronização de queries SQL, permitindo a troca de SGBD (os quais eles têm suporte) sem muitas dores de cabeça. Muitos ORM's conseguem gerar o SQL das tabelas e colocá-lo em um arquivo .sql para ajudar em futuras migrações, alguns geram até mesmo o famoso DER (Diagrama Entidade-Relacionamento), como é o caso do Prisma.

Venhamos e convenhamos, é extremamente produtivo conseguir fazer isso rapidamente com poucas configurações durante o desenvolvimento do código.

Usando SQL puro, você provavelmente teria que ter um workbench para gerar os relacionamentos do modelo lógico, e seria necessário um pouco de trabalho para obter esse diagrama.

Além disso, muitos frameworks ORM já fornecem queries prontas para consumo no desenvolvimento, como métodos como findAll, findById, create e delete, o que diminui o custo de manutenção de uma maneira surreal, dependendo do caso.

De longe, o ORM é maravilhoso para criar um MVP.
Então acho que até agora está 1 x 1.

Escalabilidade

Primeiro de tudo, o que é Escalabilidade? Ao meu ver, é a capacidade do software de continuar funcionando de forma eficiente mesmo quando a quantidade de usuários, transações ou dados aumenta significativamente.

Ok, agora vamos definir uma coisa: tanto ORM quanto SQL puro podem ser escaláveis, desde que utilizados corretamente e com a configuração adequada do banco de dados e do ambiente de execução.

No caso do ORM, ele pode facilitar a implementação de boas práticas de otimização e escalabilidade, como a criação de índices, o uso de cache de segundo nível, o lazy loading e a escrita de consultas otimizadas.

Dependendo do tipo de projeto que você está trabalhando, usar um ORM pode não ser a melhor opção se o modelo de dados for muito complexo. Isso porque, quanto mais complexo o modelo, mais difícil pode ser escrever consultas otimizadas usando o ORM. Além disso, se você estiver lidando com muitos usuários ou muitos dados ao mesmo tempo, pode ser necessário ter um controle mais detalhado do acesso ao banco de dados. Isso pode ser difícil de fazer usando um ORM, então pode ser melhor usar SQL puro ou até mesmo procedimentos armazenados no banco de dados para garantir que tudo funcione bem e que você possa controlar quem acessa o que.

Entretanto, pode ser também mais difícil manter uma aplicação com SQL puro, pois quanto mais controle você tem sobre as consultas, mais cuidado precisa ter em pontos de manutenção.
Particularmente falando, eu considero um empate, apesar dos benefícios e desvantagens do ORM, o uso de SQL puro também pode trazer uma grande dor de cabeça, dependendo da equipe que está trabalhando no projeto.

Considerações finais

Nesta postagem, foram discutidas algumas vantagens e desvantagens do uso de ORM e SQL puro. Embora muitos esperassem um vencedor claro na disputa, a verdade é que a escolha entre as tecnologias depende do contexto de cada projeto.

Se você está acostumado com o uso de SQL puro, pode ser interessante experimentar o uso de ORM na sua linguagem de programação. Não se limite a uma única tecnologia, pois quanto mais você souber extrair o melhor de diferentes ferramentas, maior será sua capacidade de adaptação a diferentes stacks.

Se me perguntarem qual tecnologia eu escolheria para criar uma aplicação agora, minha resposta seria ORM, pois já estou familiarizado com o processo de tradução do ORM para SQL, o que me permite ser mais produtivo. No entanto, se eu perceber que o projeto exige maior desempenho ou refinamento, provavelmente começaria com SQL puro. Mas, é importante lembrar que essa é apenas uma opinião pessoal e que a escolha entre as tecnologias deve ser feita com base no contexto e nas necessidades específicas de cada projeto.

Existem muitas discussões entre usar um ORM que implementa Data Mapper ou Active Record. Muitos desenvolvedores defendem um ou outro com unhas e dentes, porém, na realidade, como quase tudo na nossa área, não existe bala de prata. Se tiver a oportunidade aconselho estudar ORMs da sua linguagem que trabalha em ambos os padrões, assim você terá um conhecimento maior para escolher entre um ou outro dependendo dos requisitos do seu projeto.

ACID (Atomicidade, Consistência, Isolamento, Durabilidade

)

Em ciência da computação, ACID (acrônimo de Atomicidade, Consistência, Isolamento e Durabilidade - do inglês: Atomicity, Consistency, Isolation, Durability) é um conjunto de propriedades de transação em banco de dados.

No contexto de banco de dados, transação é uma sequência de operações de banco de dados que satisfaz as propriedades ACID e, portanto, pode ser percebida como uma operação lógica única sobre os dados. Por exemplo, uma transferência de fundos de uma conta bancária para outra, mesmo envolvendo múltiplas mudanças, como debitar uma conta e creditar outra, é uma transação única.

Em 1983, Andreas Reuter e Theo Härder inventaram o acrônimo ACID com base em trabalhos anteriores pelo cientista da computação Jim Gray que enumerou Atomicidade, Consistência e Durabilidade, mas deixou de fora o Isolamento ao caracterizar o conceito de transação. Essas quatro propriedades descrevem as principais garantias do paradigma de transação, que influenciou muitos aspectos do desenvolvimento em sistemas de banco de dados.

De acordo com Gray e Reuter, o IMS suportou as transações ACID em 1973 (embora o termo ACID tenha chegado depois).

Características

As características dessas quatro propriedades conforme definido por Reuter e Härder são as seguintes:

1. Atomicidade

Trata o trabalho como parte indivisível (atômico). A transação deve ter todas as suas operações executadas em caso de sucesso ou, em caso de falha, nenhum resultado de alguma operação refletido sobre a base de dados. Ou seja, após o término de uma transação (commit ou rollback), a base de dados não deve refletir resultados parciais da transação.

Exemplo: Em uma transferência de valores entre contas bancárias, é necessário que, da conta origem seja retirado um valor X e na conta destino seja somado o mesmo valor X. As duas operações devem ser completadas sem que qualquer erro aconteça. Em caso de algum erro, todas as alterações feitas nessa operação de transferência devem ser desfeitas.

Em projeto de banco de dados relacional, atomicidade também se refere à Primeira Forma Normal, na qual uma relação não pode conter atributos multi valorados, ou seja, o domínio de um atributo deve conter apenas valores atômicos.

2. Consistência

A execução de uma transação deve levar o banco de dados de um estado consistente a um outro estado consistente, ou seja, uma transação deve respeitar as regras de integridade dos dados (como unicidade de chaves, restrições de integridade lógica, etc.).

Todos os dados escritos no banco de dados devem ser válidos de acordo com todas as regras definidas, incluindo restrições (constraints), operações em cascata (cascade), triggers e qualquer combinação destes. Isso não garante a correção da

transação de todas as maneiras que o programador de aplicativos pode ter desejado (que é responsabilidade do código no nível do aplicativo), mas sim que qualquer erro de programação não pode resultar na violação de regras definidas.

Por exemplo, considere um banco de dados que guarde informações de clientes e que use o CPF (equivalente ao NIF em Portugal) como chave primária. Então, qualquer inserção ou alteração no banco não pode duplicar um CPF (unicidade de chaves) ou colocar um valor de CPF inválido, como o valor 000.000.000-00 (restrição de integridade lógica).

3. Isolamento

Em sistemas multi usuários, várias transações podem acessar simultaneamente o mesmo registro (ou parte do registro) no banco de dados. Por exemplo, no mesmo instante é possível que um usuário tente alterar um registro e outro usuário esteja tentando ler este mesmo registro.

O isolamento é um conjunto de técnicas que tentam evitar que transações paralelas interfiram umas nas outras, fazendo com que o resultado de várias transações em paralelo seja o mesmo resultado se as mesmas transações fossem executadas sequencialmente (uma após a outra). Operações exteriores a uma dada transação jamais verão esta transação em estados intermediários.

Fornecer isolamento é o objetivo principal do controle de concorrência. Dependendo do método de controle de concorrência (ou seja, se ele usa uma serialização rígida strict -- em oposição a relaxed --), os efeitos de uma transação incompleta podem não ser visíveis para outra transação.
Por exemplo, considere as duas seguintes transações efetuadas em um banco de dados com informações de funcionários de uma empresa:

1
: Dar um bônus de 100 reais para o empregado portador do CPF 490.120.530-30, que tem um salário de 1000 reais.
2
: Dar um bônus de 10% reais para o mesmo empregado (CPF 490.120.530-30, salário de 1000 reais).
Digamos que elas tenham a seguinte forma:
1
:

- seleciona funcionário que tenha CPF igual à 490.120.530-30;
- altera o salário somando 100 reais;
- faz o commit (confirma a gravação no banco de dados).

2
:

- seleciona funcionário que tenha CPF igual à 490.120.530-30;
- altera o salário somando 10%;
- faz o commit (confirma a gravação no banco de dados).

Se as duas transações forem executadas sequencialmente, os salário deste funcionário poderá ser:
1
e depois
2
: salário passa a ser 1100 reais e depois, somando 10% de 1100, passa a ser 1210 reais.
2
e depois
1
: salário passa a ser 1100 reais e depois, somando 100 reais, passa a ser 1200 reais.
Então, se executarmos estas duas transações em paralelo, o valor final do salário deste funcionário deve ser de 1200 ou de 1210 reais.
Se as duas transações não estiverem isoladas (uma puder ver os resultados parciais da outra), então, quando executadas em paralelo, pode ocorrer o seguinte:

- 1
- seleciona o funcionário;
- 2
- seleciona o mesmo funcionário;
- 1
- altera o salário para 1100 reais;
- 2
- altera também o salário para 1100 reais;
- 1
- grava a mudança no banco de dados;
- 2
- também grava a mudança no banco de dados.

Assim, no fim, o salário deste funcionário será igual à 1100 reais (valor gravado por
2
). Isto ocorreu neste exemplo porque deixamos as duas transações alterarem o mesmo campo (o salário) ao mesmo tempo. A propriedade de isolamento não permite que isto ocorra. Algumas técnicas de isolamento conhecidas são os bloqueios bifásicos e ordenação de marcas de tempo.

4. Durabilidade

Os efeitos de uma transação em caso de sucesso (commit) devem persistir no banco de dados mesmo em casos de quedas de energia, travamentos ou erros. Garante que os dados estarão disponíveis em definitivo. Em um banco de dados relacional, por exemplo, quando um grupo de instruções SQL é executado, os resultados precisam ser armazenados permanentemente (mesmo que o banco de dados falhe imediatamente depois). Para se defender contra a perda de energia, as transações (ou seus efeitos) devem ser registradas em uma memória não volátil.

TRANSACTIONS

O QUE SÃO TRANSACTIONS?

Uma transação simboliza uma unidade de trabalho executada dentro de um sistema de gerenciamento de banco de dados (ou sistema similar), sobre um banco de dados, e tratada de maneira coerente e confiável, independente de outras transações. Uma transação geralmente representa qualquer alteração em um banco de dados. As transações em um ambiente de banco de dados têm dois propósitos principais:

1. Fornecer unidades de trabalho confiáveis que permitam a recuperação correta de falhas e manter um banco de dados consistente mesmo em casos de falha do sistema, quando a execução é interrompida (completamente ou parcialmente) e muitas operações em um banco de dados permanecem incompletas, com estado pouco claro.
2. Fornecer isolamento entre programas que acessam um banco de dados simultaneamente. Se esse isolamento não for fornecido, os resultados dos programas possivelmente serão errôneos.

Em um sistema de gerenciamento de banco de dados, uma transação é uma unidade única de lógica ou de trabalho, às vezes composta de várias operações. Qualquer cálculo lógico feito de um modo consistente em um banco de dados é conhecido como uma transação. Um exemplo é a transferência de uma conta bancária para outra: a transação completa requer a subtração do valor a ser transferido de uma conta e a adição da mesma quantia à outra.

Uma transação de banco de dados, por definição, deve ser **atômica**, **consistente**, **isolada** e **durável**.Profissionais de banco de dados geralmente se referem a essas propriedades de transações de bancos de dados usando o acrônimo **ACID**.
As transações fornecem uma proposição "tudo ou nada", afirmando que cada unidade de trabalho executada em um banco

de dados deve ser concluída em sua totalidade ou não ter efeito algum. Além disso, o sistema deve isolar cada transação de outras transações, os resultados devem estar em conformidade com as restrições existentes no banco de dados e as transações concluídas com êxito devem ser gravadas no armazenamento durável.

O modelo relacional descreve a unidade lógica de processamento de dados como uma transaction/transação. Cada transação pode ser definida como um conjunto de operações feita em sequência. Bancos de dados relacionais – tal como o PostgreSQL – aplicam um mecanismos de locking que asseguram a integridade dessas transações.

A Parte 1 deste post foca nos conceitos básicos que garantem a execução correta de transações. Mais tarde, iremos discutir problemas de controle de concorrência na segunda parte, como também sistemas de locking, dead locks, e advisory locks.

Sintetizando

Na área de banco de dados, uma transação é uma sequência de operações num sistema de gerência de banco de dados que são tratadas como um bloco único e indivisível (atômico) durante uma recuperação de falhas e também prover isolamento entre acessos concorrentes na mesma massa de dados. Por exemplo, uma transferência de fundos de uma conta-corrente para uma conta de poupança é uma única operação do ponto de vista do cliente; contudo, dentro do sistema de banco de dados, ela consiste em várias operações. Claramente, é essencial que todas essas operações ocorram ou que, no caso de uma falha, nenhuma delas ocorra. Seria inaceitável se a conta-corrente fosse debitada, mas a conta de poupança não fosse creditada.

Essa coleção de operações deve aparecer ao usuário como uma única unidade, indivisível. A transação é indivisível, ela é executada em sua totalidade ou não é executada. Assim, se uma

transação começa a ser executada, mas falha por um motivo qualquer, quaisquer mudanças no banco de dados que a transação possa ter feito são desfeitas. Porém, é difícil garantir que esse requisito seja atendido, pois algumas mudanças no banco de dados ainda podem ser armazenadas somente nas variáveis da memória principal da transação, enquanto outras podem ter sido gravadas no banco de dados. Essa propriedade "tudo ou nada" é conhecida como atomicidade.

Além disso, como uma transação é uma unidade única, suas ações não podem ser intercaladas com outras operações do banco de dados que não participam da transação. Até mesmo um único comando SQL envolve muitos acessos separados ao banco de dados, e uma transação pode consistir em vários comandos SQL. Portanto, o sistema de banco de dados precisa tomar ações especiais para garantir que as transações operem corretamente, sem interferência de comandos de bancos de dados executando simultaneamente. Essa é a característica da propriedade chamada isolamento.

Para garantir que o sistema do banco de dados não perca uma transação completada com sucesso a partir de uma falha posterior, as ações de uma transação precisam persistir entre as falhas. Além disso, os resultados de uma transação só podem ser desfeitos por outra transação. Essa propriedade é conhecida como durabilidade.

Devido a essas três propriedades, uma transação precisa preservar a consistência do banco de dados. O modo como isso é feito é responsabilidade do programador que codifica uma transação. Essa propriedade é conhecida como consistência.

O padrão SQL define quatro níveis de isolamento de transação (Read uncommitted, Read committed, Repeatable read, Serializable) em termos de três fenômenos que devem ser evitados entre transações simultâneas. Os fenômenos não desejados são:

- dirty read (leitura suja)

A transação lê dados escritos por uma transação simultânea não efetivada (uncommitted). [1]

- nonrepeatable read (leitura que não pode ser repetida)

A transação lê novamente dados lidos anteriormente, e descobre que os dados foram alterados por outra transação (que os efetivou após ter sido feita a leitura anterior). [2]

- phantom read (leitura fantasma)

A transação executa uma segunda vez uma consulta que retorna um conjunto de linhas que satisfazem uma determinada condição de procura, e descobre que o conjunto de linhas que satisfazem a condição é diferente por causa de uma outra transação efetivada recentemente.

Os quatro níveis de isolamento de transação, e seus comportamentos correspondentes, estão descritos na tabela abaixo:

Níveis de isolamento da transação no SQL

Nível de isolamento	Dirty Read	Nonrepeatable Read	Phantom Read
Read uncommitted	Possível	Possível	Possível
Read committed	Impossível	Possível	Possível
Repeatable read	Impossível	Impossível	Possível
Serializable	Impossível	Impossível	Impossível

A partir de onde é Importante

A partir do momento em que a aplicabilidade de transações em Banco de Dados torna possível que haja uma transação limpa sem

complicações eminentes por conta dos cuidados com cada procedimento que se faz necessário para aplicações que são de suma importância em nosso mundo hoje.

Façamos uma análise

Suponha que um usuário apresente a seguinte atualização a um banco de dados usado para processar compensação de cheques bancários. Debite $100,00 à conta corrente do cliente N Credite $100,00 à conta corrente do cliente M Após manipulação pelo processador de consultas, teríamos algo como a seguinte seqüência de ações elementares:

- A1 = (LEIA,CONTA_CORRENTE[A].SALDO,X)
- A2 = (COMPUTE, W:=X-100)
- A3 = (ESCREVA,CONTA_CORRENTE[A].SALDO,W)
- A4 = (LEIA,CONTA_CORRENTE[B].SALDO,X)
- A2 = (COMPUTE, W:=X+100)
- A6 = (ESCREVA,CONTA_CORRENTE[B].SALDO,W)
-

[[onde A e B são os elementos de CONTA_CORRENTE correspondentes às tuplas de CONTAS com NUMERO_CONTA = N e NUMERO_CONTA = M, respectivamente.]]

Por alguma razão qualquer (falta de energia, falhas no equipamento ou programas, etc.)., a atividade do banco de dados é interrompida após a execução da ação elementar A3. Ao retornar à atividade, teríamos debitado $100,00 à conta de A e ainda não creditado valor algum à conta de B. Isto colocaria o banco de dados num estado inconsistente, pois $100,00 teriam simplesmente "desaparecido". Seria muito conveniente se pudéssemos assegurar que a seqüência de ações elementares acima tivesse a propriedade de que "ou todas as ações elementares executam com sucesso ou nenhuma delas é executada". Em outras palavras, gostaríamos de estender o conceito de atomicidade de modo a envolver agora também sequências de ações elementares.

A atomicidade de seqüência de ações elementares poderia também ser violada por interferência de outras ações executadas concorrentemente (exemplos serão dados no Capítulo 8).

A noção de transação é introduzida para forçar o sistema a executar uma seqüência de ações elementares como se fosse uma unidade atômica, sem interferência externa. A nível lógico, uma transação é definida através de um programa em uma linguagem de alto nível, contendo comandos da LMD embebidos, e iniciado e terminado pelos comandos COMEÇO-DE-TRANSAÇÃO e FIM-DE-TRANSAÇÃO. Na sua forma mais simples, uma transação contém apenas um comando da LMD. Exige-se do usuário que codifique as transações de tal forma que quando executadas sozinhas:

- T1. sempre terminem;
- T2. preservem a consistência do banco de dados.

Exige-se do SGBD, por sua vez, que a cada invocação de uma transação T:

- S1. a transação T seja executada por completo;
- S2. a execução da transação T se dê sem interferência de outras transações que porventura estejam sendo executadas concorrentemente.

Lembremos que a cada transação T corresponde uma seqüência de ações elementares sobre os objetos físicos. O primeiro requisito, S1, garante então que o efeito líquido, isto é, aquele que será tornado público após o término da chamada a T, refletirá o fato de que todas ou nenhuma das ações elementares de T foram executadas.

Note que isto não implica que cada uma das ações de T deva ser ativada apenas uma única vez. Para ver isto, suponha que existam, no repertório de ações elementares do nó onde T foi executada, as ações elementares A1 ,…, An onde Ai indica a ação elementar de efeito exatamente contrário à Ai . Em outras palavras, Ai devolve o banco de dados ao estado anterior à execução de Ai . Retornando ao exemplo anterior sobre compensação bancária, é fácil ver que A1 , como uma

operação de "leitura", não altera o estado do banco. Portanto, seria válido colocar A1 = (NULO), isto é, à A1 não corresponde operação alguma. Idem, A2 = (NULO). Continuando, A3 = (ESCREVA, CONTA_CORRENTE[A].SALDO, X) pois a variável X detém o conteúdo inicial de CONTA_CORRENTE[A].SALDO até o término da transação. Note que isto restaura o valor do objeto CONTA_CORRENTE[A] ao valor que apresentava antes da execução de A3. Podemos definir A4, A5 e A6 analogamente. Dentro deste espírito, se E = (A1, A2, A3, A4) for a seqüência de ações elementares gerada por uma execução seqüencial de T, então cada uma das seqüências de ações elementares abaixo, quando ativadas pelo sistema, satisfazem ao requisito contido no item (1) acima. A1 , A2 , A3 , A4 A1 , A2 , A2 , A2 , A3 , A4 , A4 , A4 A1 , A1 , A1 , A1 , A1 , A2 , A3 , A3 , A2 , A1 , A1 , A2 , A3 , A4 Na realidade, embora as ações elementares Ai pareçam um tanto obtusas no momento, desempenharão um papel crucial nas idéias a serem desenvolvidas mais adiante.

O leitor atento já deve ter percebido isto. Voltando ao exemplo da compensação bancária, caso o sistema falhe após a execução das ações elementares A1 , A2 e A3 , e supondo que os objetos envolvidos residam em memória secundária, ao retornar à atividade o banco de dados estaria em um estado refletindo uma execução parcial de T.

Neste ponto, após executada as ações U = (A3 , A2 , A1) o banco de dados recairia novamente em um estado que não reflete nenhuma ação de T, podendo retomar suas atividades normais. O segundo requisito, S2, não exclui a possibilidade de intercalar ações elementares de transações diferentes sobre o mesmo banco de dados local, ou de executar paralelamente ações elementares da mesma transação ou de transações diferentes sobre bancos de dados locais distintos. No entanto, o requisito S2 exige que algum controle seja exercido para que o nível de concorrência permitido não destrua a idéia de atomicidade da execução das transações.

Executando Transações: Início, Migração E Término

Recordemos inicialmente a arquitetura para SGBDDs introduzida na Seção 1.1.3. Em um primeiro nível, esta arquitetura divide o SGBDD em uma coleção de SGBDs locais interligados pelo SGBD global. Cada nó da rede possui uma cópia do SGBD global. Este, por sua vez, é dividido em três grandes componentes:

- 1. diretório de dados global (DDG): contém descrições dos objetos lógicos e físicos e dos mapeamentos entre estes;
- 2. gerente de transações (GT): interpreta e controla o processamento de consultas e transações submetidas ao sistema;
- 3. gerente de dados (GD): interface com o SGBD local, fazendo as traduções necessárias no caso de sistemas heterogêneos.

O propósito desta seção será apresentar em mais detalhe como um GT processa transações.

- Em uma primeira fase:

coordenador: envia, a todos os nós participantes, mensagens na forma PREPARE-SE. cada nó participante

- 1. ao receber uma mensagem de PREPARE-SE vinda do coordenador, tenta registrar em memória estável, isto é, de forma que sobreviva à eventuais falhas do sistema, os efeitos locais da transação
- 2. a seguir, envia ao coordenador mensagens na forma PREPARADO caso tenha conseguido o registro em memória estável, ou na forma IMPOSSIBILITADO em caso contrário

- Numa segunda fase:

coordenador

- 1. envia a todos os nós participantes a mensagem CONFIRME, caso deseje confirmar a transação e todos os nós participantes tenham remetido a mensagem PREPARADO na primeira fase;

    - 2. em qualquer outro caso, o coordenador envia a mensagem CANCELE a todos

os nós participantes cada nó participante: ao receber o veredito do coordenador, procede de acordo com este.

TRANSACTIONS

Uma transação/transaction é um conjunto de operações que podem atualizar, deletar, inserir e retornar dados. Essas operações podem ser explicitamente agrupadas em um bloco de transação ao usarmos as declarações BEGIN e END. Uma transação ocorre com sucesso apenas se todas as operações que compõe a transação forem bem sucedidas. Se uma operação dentro da transação falhar, o efeito das demais operações parcialmente concluídas pode ser desfeito.

Para definir o início de uma transação explicitamente, usamos a declaração BEGIN. Para declarar o fim de uma transação, declaramos END ou COMMIT para concretizar as operações ou ROLLBACK para anulá-las. O exemplo a seguir mostra como executar uma operação em SQL dentro de uma transação.

```
BEGIN;
CREATE TABLE empregado (empregado_id serial primary key, nome text, salario numeric);
COMMIT;
```

Uma das vantagens em se usar transactions, além de assegurar a integridade dos dados, é a de providenciar uma maneira fácil de desfazer alterações no banco de dados em, por exemplo, ambientes de teste e de desenvolvimento. Um bloco de transação pode conter declarações de **SAVEPOINT** para poder manipular os dados de forma interativa. O **SAVEPOINT** é um ponto de referência dentro da transação onde o estado dos dados é salvo. **SAVEPOINTS** desfazem partes da transação ao invés de anulá-la por completo. O exemplo a seguir mostra como o SAVEPOINT pode ser utilizado:

```
BEGIN;
UPDATE empregado set salario = salario * 1.1;
SAVEPOINT aumento_salario;
UPDATE empregado set salario = salario + 500 WHERE nome
='José';ROLLBACK to aumento_salario;
COMMIT;
```

Neste exemplo, o comando

```
UPDATE empregado set salario = salario * 1.1;
```

é perpetrado (COMMIT), ao passo que o comando

```
UPDATE empregado set salario = salario + 500 WHERE nome
='José';
```

é revertido (**ROLLBACK**), não alterando os dados de fato. Isso ocorre pois a transação recuou para o estado de dados presente no **SAVEPOINT** aumento_salario.

OBS: Todos os comandos executados no PostgreSQL são transacionais, mesmo que você não declare um bloco de transação explicitamente. Caso você execute, por exemplo, 30 comandos UPDATE sem inserí-los dentro de um bloco de transação explicitamente, cada um dos comandos é executado em um bloco de transação separado e independente.

N+1 PROBLEM

Mas o que é realmente esse problema?

A melhor forma de explicar este problema é com um exemplo. Imagine que você tem uma tabela Pessoa e uma tabela Endereco. Cada pessoa tem vários endereços, consolidando uma relação de um para muitos (1-N).

E agora você deseja pegar os endereços de várias pessoas. Normalmente, vemos a seguinte consulta utilizando o ORM de sua preferência (vou usar a notação JPQL do JPA):

```
public List<Pessoa> consultarPessoas() {
     String jpql = "select * from Pessoa";
     return em.createQuery(jpql).getResultList();
}
```

E, em seguida, você itera por cada Pessoa para pegar os seus endereços:

```
List<Pessoa> pessoas = consultarPessoas(): for(Pessoa pessoa : pessoas) {
     List<Endereco> enderecos = pessoa.getEnderecos();
}
```

Imaginando um LAZY entre pessoa e endereços, teremos o seguinte SQL para cada pessoa ao chamar o método **pessoa.getEnderecos():**

```
SELECT * from Endereco where pessoa_id = :id;
```

, por que ele acontece,

O problema ocorre porque para pegar os endereços das pessoas você pega primeiro a pessoa e depois busca os endereços de cada. Imaginando que a consulta anterior nos retornou 5 pessoas, a quantidade de SQLs gerados será algo assim:

```
SELECT * FROM pessoa SELECT * from endereco
WHERE pessoa_id = 1;
SELECT * FROM endereco WHERE pessoa_id = 2;
SELECT * FROM endereco WHERE pessoa_id = 3;
SELECT * FROM endereco WHERE pessoa_id = 4;
SELECT * FROM endereco WHERE pessoa_id = 5;
```

Ou seja, 1 select de pessoa com N select para endereços, o famoso N + 1 .

Quais suas principais causas?

Normalmente ela é causada pelo uso inadequado dos ORMs. É preciso entender o que o ORM faz por trás dos bastidores. Embora eles estejam aí para facilitar nossa vida, eles precisam ser usados com sabedoria. Por serem muito permissivos de forma geral, resultados inesperados podem ser causados no mal uso da ferramenta.

E como, na teoria, resolvê-los?

No exemplo que dei anteriormente, seu objetivo era pegar os endereços de várias pessoas. Se forem os endereços de todas as pessoas do banco de dados, você precisa apenas fazer:

```
SELECT * FROM Endereco
```

Mas se quiser aplicar um filtro para trazer aquelas 5 pessoas, isto pode ser feito evitando aquelas várias consultas com um JPQL diferente, partindo da tabela Endereco também:

```
SELECT * FROM Endereco WHERE pessoa_id IN (1,2,3,4,5);
```

Já ouvi também que para resolver, é só praticar o eager loading. Mas até que ponto ele é benéfico e capaz de resolver esse problema?
O EAGER loading é uma alternativa, pois o SQL gerado seria algo assim:
select p.id, p.nome, end.id, end.rua, end.pessoa_id from pessoa p JOIN endereco end ON end.pessoa_id = p.id

Contudo, o EAGER é um problema se adicionado entre o relacionamento de Pessoa e Endereço no seu ORM, pois toda vez que você buscar uma pessoa, os endereços também virão juntos. Acredite, você não quer isto como comportamento padrão do seu sistema. Os principais problemas de performance que vi em aplicações que envolviam o uso de algum ORM eram causados por isto.

Alguns ORMs tem a alternativa de usar o FETCH opcionalmente em uma consulta, assim você pode "ligar" o EAGER quando quiser. No JPQL, ficaria assim:

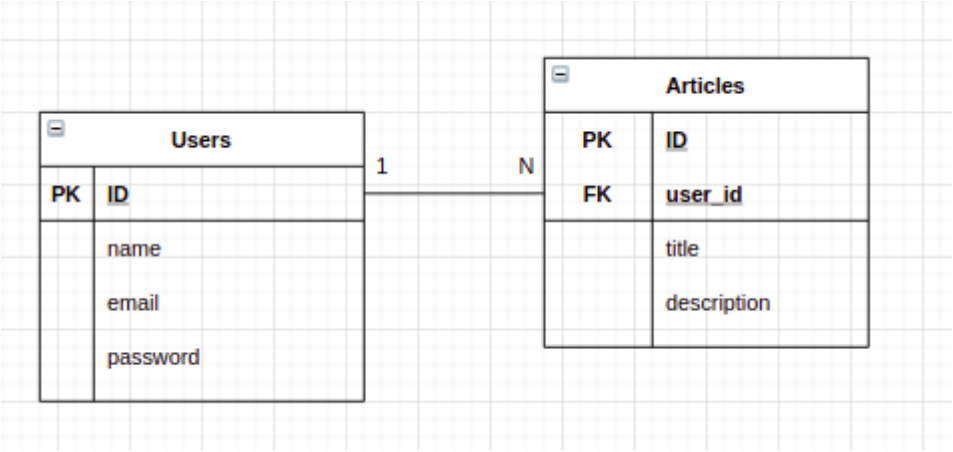
```
SELECT * FROM pessoa JOIN FETCH pessoa.enderecos;
```

Resultando no mesmo SQL que citei anteriormente.
Porém, existe uma séria limitação com FETCH e EAGER se você tentar aplicar algum tipo de paginação na consulta. Usando EAGER ou FETCH isto não é possível de ser aplicado no próprio SQL gerado, e para ter uma consulta com este mesmo efeito (trazer pessoas e endereços na mesma consulta) você precisará apelar uma consulta nativa e/ou usar funções nativas (como o DENSE_RANK) do banco de dados.

O problema N+1 é um anti-pattern, conhecido por ser uma maneira ineficiente de realizar consultas numa base com um número relativamente grande de dados.
Neste artigo irei explicar o comportamento deste problema, como identifica-lo e como corrigi-lo. Mantendo o desempenho da suas aplicações eficiente.

Como se comporta?

Vamos imaginar que temos numa base de dados duas tabelas, uma de usuários e outra de artigos.



A tabela de usuários armazena as informações de cada usuário registrado no sistema e a tabela de artigos armazena as informações dos artigos de cada usuário. Logo temos uma relação de 1 para N, sendo o N o número de artigos que cada usuário escreveu.

Imagine que você precise desenvolver um dashboard onde será exibido dados de todos os usuários e os títulos de todos os artigos de cada um deles. O que provavelmente você pensaria em fazer de primeira seria:
1°- Buscar pelas informações de todos os usuários:

```
SELECT ID, name, email FROM Users;
```

2°- Em seguida consultar o título de todos os artigos de cada usuário:

```
SELECT title FROM Articles WHERE user_id = id_usuario;
```

Agora pense que temos 500 usuários registrados na nossa base. As consultas serão executadas da seguinte maneira:

```
SELECT ID, name, email FROM Users;
SELECT title FROM Articles WHERE user_id = 1;
SELECT title FROM Articles WHERE user_id = 2;
SELECT title FROM Articles WHERE user_id = 3;
SELECT title FROM Articles WHERE user_id = 4;
...
SELECT title FROM Articles WHERE user_id = 500;
```

O que podemos reparar é que no final executamos 501 consultas ao nosso banco de dados, por isso o nome N+1, sendo N o número de usuários.

Nesse exemplo acima temos poucas colunas nas nossas tabelas, mas muitas vezes, em sistemas empresariais de médio à grande porte, o número de colunas numa tabela pode ser bem maior. Isso faz com que o tempo que nossas consultas irão levar e o tempo de resposta da nossa aplicação aumente. O que afinal de contas, não é nada bom.

Como identificar?

Na grande maioria das vezes, quando temos uma consulta dentro de um loop temos o problema de N+1. Trazendo um pouco mais para o dia a dia do desenvolvedor, abaixo deixo um pseudocódigo exemplificando este cenário:

```
var db_connection = new DB("data to conect");

var query = "SELECT ID, name, email FROM Users;";

var prepare_query = db_connection->prepare(query);

var users = prepare_query->execute();

foreach (user of users) {
    var article_query = `SELECT title FROM Articles WHERE user_id = ${user.ID}`;

    var prepare_article_query = db_connection->prepare(article_query);

    var articles_from_user = prepare_article_query->execute();
}
```

Como podemos notar, faremos uma primeira consulta que nos trás todos os usuários. Logo após, faremos N consultas, sendo N o número de usuários e para cada iremos trazer os artigos relacionados a ele.

Como resolver?

Agora vem a parte boa, vamos transformar este número de 501 consultas para apenas 2!
Primeiramente, iremos continuar fazendo a consulta que nos trás os usuários:

SELECT ID, name, email FROM Users;

Agora, iremos utilizar estas informações que temos para fazer a nossa segunda consulta. Nessa precisaremos antes de tudo iterar sobre nosso array de usuários para termos apenas um novo array com todos os IDs. Ficando da seguinte maneira :

```
SELECT title FROM Articles WHERE user_id IN (array_of_users_id);
// SELECT title FROM Articles WHERE user_id IN (1,2,3,4,...,500);
```

Alterando o pseudocódigo que montamos anteriormente, ele segueria a seguinte lógica:

```
var db_connection = new DB("data to connect");

var query = "SELECT ID, name, email FROM Users;";

var prepare_query = db_connection->prepare(query);

var users = prepare_query->execute();

var array_of_users_id = map(users, "id");

var articles_query = `SELECT title FROM Articles WHERE user_id IN (${...array_of_users_id});`;

var prepare_articles_query = db_connection->prepare(articles_query);

var articles_from_users = prepare_articles_query->execute();
```

Assim , o que fizemos de modificação foi apenas aproveitar os dados retornados dos usuários para criar um novo array, mas agora apenas com o ID de cada usuário, e realizar uma consulta no banco por todos os artigos que possuem relação com um dos IDs informados no operardor IN. Totalizando apenas 2 consultas feitas ao banco.

Mas… e se usarmos JOIN?

Se você possui experiência com SQL, perceberá que podemos ao invés de usar o operador IN, realizar um JOIN e com isso apenas precisariamos de 1 consulta.

O número de consultas pode afetar no desempenho da nossa aplicação, mas isso não significa que por ser 1 consulta será mais eficiente.

```
SELECT Users.ID, Users.nome, Articles.title FROM Users INNER JOIN Articles ON Articles.user_id = Users.ID;
```

Por exemplo se cada usuário apenas tivesse um artigo escrito, o JOIN seria mais eficiente. Mas se cada usuário tiver em média 5 artigos, teremos muitos dados repetidos trafegando entre o banco de dados e a aplicação, o que pode consumir tempo e memória.

Conclusão

No final das contas, a modificação que fizemos para alterar o número de consultas na nossa base foi bem simples, mas muito poderosa.
Com ela tivemos um ganho de desempenho muito alto, deixando o tempo de resposta da nossa aplicação menor e consequentemente fazendo nosso cliente final muito mais feliz.

Normalização de banco de dados

Normalização de banco de dados é um conjunto de regras que visa, principalmente, a organização de um projeto de banco de dados para reduzir a redundância de dados, aumentar a integridade de dados e o desempenho. Para normalizar o banco de dados, deve-se examinar as colunas (atributos) de uma entidade e as relações entre entidades (tabelas), com o objetivo de se evitar anomalias observadas na inclusão, exclusão e alteração de registros.

Atualmente, muitos sistemas de informação ainda não utilizam banco de dados relacionais, sendo esses chamados de sistemas legados. Os dados desses sistemas são armazenados em arquivos de linguagens de terceira geração, como COBOL ou BASIC, ou então, em banco de dados da era pré-relacional.

Panorâmica informal

Para adequar o banco de dados, é necessário avaliar com base em cinco regras, que recebem o nome de formas normais. Essas correspondem a um conjunto de regras de simplificação e adequação de tabelas. Diz-se que a tabela do banco de dados relacional está numa certa forma normal quando satisfaz as condições exigentes. O trabalho original de Edgar F. Codd, definiu três dessas formas, mas existem hoje outras formas normais geralmente aceitas. Damos aqui uma curta panorâmica informal das mais comuns. Cada forma normal listada abaixo representa uma condição mais forte das que a precedem na lista. Para a maioria dos efeitos práticos, considera-se que as bases de dados estão normalizadas se aderirem à terceira forma normal.

Inicialmente, são definidos todos os atributos do documento, que estão relacionados a uma entidade principal, atribuindo uma chave primária. Feito isso, partimos para a análise do documento de acordo com as formas normais a seguir:

- Primeira Forma Normal (ou 1FN). Nesta forma os atributos precisam ser atômicos, o que significa que as tabelas não podem ter valores repetidos e nem atributos possuindo mais de um valor. Exemplo: CLIENTE = {ID + ENDEREÇO + TELEFONES}. Porém, uma pessoa poderá ter mais de um número de telefone, sendo assim o atributo "TELEFONES" é multivalorado. Para normalizar, é necessário:
    1. Identificar a chave primária e também a coluna que possui dados repetidos (nesse exemplo "TELEFONES") e removê-los;
    2. Construir uma outra tabela com o atributo em questão, no caso "TELEFONES". Mas não se esquecendo de fazer uma relação entre as duas tabelas: CLIENTE = {ID + ENDEREÇO} e TELEFONE (nova tabela) = {CLIENTE_ID (chave estrangeira) + TELEFONE}.
- Segunda Forma Normal (ou 2FN). Primeiramente, para estar na 2FN é preciso estar também na 1FN. 2FN define que os atributos normais, ou seja, os não chave, devem depender

unicamente da chave primária da tabela. Assim como as colunas da tabela que não são dependentes dessa chave devem ser removidas da tabela principal e cria-se uma nova tabela utilizando esses dados. Exemplo: PROFESSOR_CURSO = {ID_PROF + ID_CURSO + SALARIO + DESCRICAO_CURSO} Como podemos observar, o atributo "DESCRICAO_CURSO" não depende unicamente da chave primária "ID_PROF", mas sim somente da chave "ID_CURSO". Para normalizar, é necessário:

1. Identificar os dados não dependentes da chave primária (nesse exemplo "DESCRICAO_CURSO") e removê-los;
2. Construir uma nova tabela com os dados em questão: PROFESSOR_CURSO = {ID_PROF + ID_CURSO + SALARIO} e CURSOS (nova tabela) = {ID_CURSO + DESCRICAO_CURSO}.

- Terceira Forma Normal (ou 3FN). Assim como para estar na 2FN é preciso estar na 1FN, para estar na 3FN é preciso estar também na 2FN. 3FN define que todos os atributos dessa tabela devem ser funcionalmente independentes uns dos outros, ao mesmo tempo que devem ser dependentes exclusivamente da chave primária da tabela. 3FN foi projetada para melhorar o desempenho de processamento dos banco de dados e minimizar os custos de armazenamento. Exemplo: FUNCIONARIO = {ID + NOME + VALOR_SALARIO + VALOR_FGTS}. Como sabemos o valor do FGTS é proporcional ao salário, logo o atributo normal "VALOR_FGTS" é dependente do também atributo normal "VALOR_SALARIO". Para normalizar, é necessário:
  1. Identificar os dados dependentes de outros (nesse exemplo "VALOR_FGTS");
  2. Removê-los da tabela. Esses atributos poderiam ser definitivamente excluídos -- e deixando para a camada de negócio a responsabilidade pelo seu cálculo -- ou até ser movidos para uma nova tabela e referenciar a principal ("FUNCIONARIO").
- Forma Normal de Boyce-Codd (ou BCNF) requer que não exista nenhuma dependência funcional não trivial de atributos em algo mais do que um superconjunto de uma

chave candidata. Neste estágio, todos os atributos são dependentes de uma chave, de uma chave inteira e de nada mais que uma chave (excluindo dependências triviais, como $A \rightarrow A$);
- Quarta Forma Normal (ou 4FN) requer que não exista nenhuma dependência multi-valorada não trivial de conjuntos de atributo em algo mais de que um superconjunto de uma chave candidata;
- Quinta Forma Normal (ou 5FN ou PJ/NF) requer que não exista dependências de joins (associações) não triviais que não venham de restrições chave;
- Domain-Key Normal Form (ou DK/NF) requer que todas as restrições sigam os domínios e restrições chave.

Visão Formal

Antes de falar sobre normalização, é necessário utilizar alguns termos a partir do modelo relacional e defini-los na teoria de conjuntos. Estas definições muitas vezes serão simplificações de seus significados originais, uma vez que somente alguns aspectos do modelo relacional são levados em consideração na normalização.

As notações básicas utilizadas no modelo relacional são nomes de relacionamentos e nomes de atributos, representados por cadeias de caracteres tais como Pessoas e Nomes; geralmente são utilizadas variáveis como r, s, t, … e a, b, c para o conjunto dados definido sobre eles. Outra notação básica é o conjunto de valores atômicos que contém valores tais como números e cadeias de caracteres.

A primeira definição de interesse é a noção de tupla, que formaliza a noção de linha ou registro em uma tabela:
Def. Uma tupla é uma função parcial de nomes de atributos para valores atômicos.

Def. Um cabeçalho é um conjunto finito de nomes de atributos.
Def. A projeção de uma tupla t em um conjunto finito de atributos A é $t[A] = \{ (a, v) : (a, v) \in t, a \in A \}$.
A próxima definição é a de relação na qual formaliza-se o teor de uma tabela como ele é definido no modelo relacional.

Def. Uma relação é uma tupla (H, B) sendo H um cabeçalho e B o corpo, um conjunto de tuplas em que possuem todas o domínio H. Como uma relação corresponde definitivamente com aquela que é usualmente chamada de extensão de um predicado em lógica de primeira ordem exceto que aqui nós identificamos os locais no predicado com nomes de atributos. Geralmente no modelo relacional um esquema de banco de dados é dito consistir-se de um conjunto de nomes relação, os cabeçalhos que são associados com esses nomes e as restrições que devem manter toda instância do esquema de banco de dados. Para normalização nós nos concentraremos nas restrições que indicam relações individuais, isto é, as restrições relacionais. O propósito destas restrições é descrever o universo relacional, ou seja, o conjunto de todas as relações que são permitidas para serem associadas com certos nomes de relação.

Def. Um universo relacional U sobre um cabeçalho H é um conjunto não vazio de relações com o cabeçalho H.
Def. Um esquema relacional (H, C) consiste de um cabeçalho H e um predicado C(R) que é definido por todas as relações R com o cabeçalho H.

Def. Uma relação satisfaz o esquema relacional (H, C) se possuir o cabeçalho H e satisfizer C.
Restrições Chave e Dependências Funcionais
A restrição relacional mais importante é a restrição de Chave. Ela relaciona cada registro (tupla) a um (ou mais) valor índice.

Def. Uma Chave é um atributo que identifica um registro (tupla).

Objetivos de normalização

Um objetivo básico da primeira forma normal, definida por Codd em 1970, era permitir dados serem questionados e manipulados usando uma "sub-linguagem de dados universal" atrelada à lógica de primeira ordem. Questionando e manipulando dados em uma estrutura de dados não normalizada, como a seguinte representação não-1NF de transações de clientes de cartão de crédito, envolve mais complexidade do que seria realmente necessário:

<table>
<tr><th>Cliente</th><th>Cliente ID</th><th>Transação</th></tr>
<tr><td>João</td><td>1</td><td>
<table>
<tr><th>Tr. ID</th><th>Data</th><th>Valor</th></tr>
<tr><td>12890</td><td>14/out/2003</td><td>-87</td></tr>
<tr><td>12904</td><td>15/out/2003</td><td>-50</td></tr>
</table>
</td></tr>
<tr><td>Wilson</td><td>2</td><td>
<table>
<tr><th>Tr. ID</th><th>Data</th><th>Valor</th></tr>
<tr><td>12898</td><td>14/out/2003</td><td>-21</td></tr>
</table>
</td></tr>
<tr><td>Márcio</td><td>3</td><td>
<table>
<tr><th>Tr. ID</th><th>Data</th><th>Valor</th></tr>
<tr><td>12907</td><td>15/out/2003</td><td>-18</td></tr>
<tr><td>14920</td><td>20/nov/2003</td><td>-70</td></tr>
<tr><td>15003</td><td>27/nov/2003</td><td>-60</td></tr>
</table>
</td></tr>
</table>

Note que a tabela é composta pelas colunas cliente e transação, sendo que essa última contém 3 informações: identificador, data e valor.
Para cada cliente corresponde um grupo repetitivo de transações.

A análise automatizada de transação envolve dois estágios:

1. Desempacotar um ou mais grupos de clientes de transações permitindo transações individuais serem agrupadas para exame, e
2. Derivar o resultado de uma consulta em resultados do primeiro estágio.

Por exemplo, para encontrar a soma monetária de todas as transações que ocorreram em outubro de 2003 para todos os clientes, o sistema necessitaria saber primeiro que precisa desempacotar o grupo de transações para cada cliente, então somar a quantidade de todas as transações de outubro de 2003.
Um das visões mais importantes de Codd foi que a complexidade desta estrutura poderia ser removida completamente, levando a um poder e flexibilidade muito maior na forma de efetuar consultas. A normalização equivalente da estrutura acima seria assim:

Cliente	Cliente ID
João	1
Wilson	2
Márcio	3

Cliente ID	Tr. ID	Data	Valor
1	12890	14/out/2003	-87
1	12904	15/out/2003	-50
2	12898	14/out//2003	-21
3	12907	15/out/2003	-18
3	14920	20/nov/2003	-70
3	15003	27/nov/2003	-60

Nessa nova estrutura, as chaves são {Cliente} e {Cliente ID} na primeira tabela e {Cliente ID, Tr. ID} na segunda.
Agora cada linha representa uma transação individual, e um SGBD pode obter a resposta, simplesmente encontrando todas as linhas com data de outubro, somando então os valores.

Formas Normais

Primeira Forma Normal
Definição pelo Osório
'Uma tabela está na 1FN, se e somente se, todos os valores das colunas da tabela forem atómicos'
Assim, podemos dizer que os relacionamentos, como definidos acima, estão necessariamente na 1FN. Uma relação está na 1FN quando todos os atributos da relação estiverem baseados em um domínio simples, não contendo grupos ou valores repetidos.

Passagem à 1FN

1. Encontre a chave primária da tabela;
2. Fique ciente de quais são as colunas da tabela que apresentam dados repetidos para que sejam removidas;
3. Crie uma tabela para esses dados repetidos, com a chave primária da anterior;
4. Por fim, estabeleça relação entre a nova tabela e a principal.

Outra forma de identificar se a tabela não está na 1FN é verificando se existe tabela aninhadas, ou seja, mais de um registro para uma chave primária.

Exemplo
Observe a seguinte tabela:
PEDIDOS = {COD_PEDIDO + CLIENTE + VENDEDOR + ATENDENTE + PRODUTO + QUANT + VALOR}
Considere um pedido como o número 00001, para este se observarmos o formulário em papel teremos muitos campos a

considerar, contudo usaremos apenas alguns para facilitar o entendimento.
Neste momento devemos idealizar o pedido número 00001 e efetuar os seguintes testes:

{COD_PEDIDO | CLIENTE | VENDEDOR | ATENDENTE | PRODUTO | QUANT | VALOR}
{00001 | "ANDRÉ" | "MARCO" | "JOAO" | "TENIS " | "1" | "50.00"}
{00001 | "ANDRÉ" | "MARCO" | "JOAO" | "SANDALIA" | "2" | "80.00"}
{00001 | "ANDRÉ" | "MARCO" | "JOAO" | "CARTEIRA" | "1" | "35.00"}

Observe que para os dados do pedido 00001 lançados acima, apenas os atributos que estão em negrito SÃO ÚNICOS, pois não se diferem. Os demais atributos mudam, não cumprindo a 1FN onde os atributos devem ser atômicos, quer dizer únicos.
Para testarmos um dos atributos e ter certeza que este é atômico, podemos efetuar uma pergunta conforme o exemplo abaixo:

Podemos ter outro cliente para o pedido 00001? Não. Podemos ter apenas 1 cliente por pedido, sendo assim este atributo é atômico único para 1 pedido.

Podemos ter outro vendedor para o pedido 00001? Não. Podemos ter apenas 1 vendedor por pedido.
Podemos ter outro produto para o pedido 00001? Sim. Podemos ter vários produtos para um pedido, sendo assim, os campos aninhados devem ser extraídos para outra tabela.

Problemas

- Redundância;
- Anomalias de Atual.

Segunda Forma Normal
Definição pelo Osório

Uma relação está na 2FN se, e somente se, estiver na 1FN e cada atributo não-chave for dependente da chave primária inteira, isto é, cada atributo não-chave não poderá ser dependente de apenas parte da chave.

No caso de tabelas com chave primária composta, se um atributo depende apenas de uma parte da chave primária, então esse atributo deve ser colocado em outra tabela.

Passagem à 2FN

1. Geração de novas tabelas com DFs (Dependências Funcionais) completas.
2. Análise de dependências funcionais:
    - tipo e descrição dependem de codp;
    - nome, categ e salário dependem de code;
    - data_início e tempo_aloc dependem de toda a chave.

Conclusões

- Maior independência de dados;
- Redundâncias e anomalias: dependências funcionais indirectas.
-

Terceira Forma Normal

Definição pelo Osório

Uma relação R está na 3FN se ela estiver na 2FN e cada atributo não-chave de R não possuir dependência transitiva, para cada chave candidata de R. Todos os atributos dessa tabela devem ser independentes uns dos outros, ao mesmo tempo que devem ser dependentes exclusivamente da chave primária da tabela.

Exemplo ilustrativo

A tabela a seguir não está na Terceira Forma Normal porque a coluna Total é dependente, ou é resultado, da multiplicação das colunas Preço e Quantidade, ou seja, a coluna total tem dependência transitiva de colunas que não fazem parte da chave primária, ou mesmo candidata da tabela. Para que essa tabela passe à Terceira FN o campo Total deverá ser eliminado, a fim

de que nenhuma coluna tenha dependência de qualquer outra que não seja exclusivamente chave.

Itens do pedido				
Pedido	**Item**	**Preço**	**Quantidade**	**Total**
15	102	9,25	2	18,5
15	132	1,3	5	6,5

Aqui, após o atributo/coluna Total ser excluído da tabela, ela já na 3ª Forma Normal. Esse atributo pode ser movido para outra tabela referenciando a antiga.

Itens do pedido			
Pedido	Item	Preço	Quantidade
15	102	9,25	2
15	132	1,3	5

Passagem à 3FN

- Para estar na 3FN precisa estar na 2FN;
- Geração de novas tabelas com DF diretas;
- Análise de dependências funcionais entre atributos não chave;
- Verificar a dependência exclusiva da chave primária;
- Entidades na 3FN também não podem conter atributos que sejam resultados de algum cálculo de outro atributo.

Conclusões

- Maior independência de dados;
- 3FN gera representações lógicas finais na maioria das vezes;
- Redundâncias e anomalias: dependências funcionais.

Quarta Forma Normal

Definição

Uma tabela está na 4FN, se e somente se, estiver na 3FN e não existirem dependências multivaloradas.

Exemplo (base de dados sobre livros)

Relação não normalizada: Livros(nrol, (autor), título, (assunto), editora, cid_edit, ano_public)

1FN: Livros(nrol, autor, assunto, título, editora, cid_edit, ano_public)

2FN: Livros(nrol, título, editora, cid_edit, ano_public)
AutAssLiv(nrol, autor, assunto)

3FN: Livros(nrol, título, editora, ano_public)
Editoras(editora, cid_edit)
AutAssLiv(nrol, autor, assunto)

Na 3FN, a base de dados ainda apresenta os seguintes problemas:

- Redundância para representar todas as informações;
- Representação não-uniforme (repete alguns elementos ou posições nulas).

Passagem à 4FN

- Geração de novas tabelas, eliminando dependências multivaloradas;
- Análise de dependências multivaloradas entre atributos:
  - autor, assunto → Dependência multivalorada de nrol.

Resultado
Livros(nrol, título, editora, ano_public)
Editoras(editora, cid_edit)
AutLiv(nrol, autor)
AssLiv(nrol, assunto)

Quinta Forma Normal
Está ligada à noção de dependência de junção.

- Se uma relação é decomposta em várias relações e a reconstrução não é possível pela junção das outras relações, dizemos que existe uma dependência de junção.
- Existem tabelas na 4FN que não podem ser divididas em duas relações sem que se altere os dados originais.

Exemplo: Sejam as relações R1(CodEmp, CodPrj) e R2(CodEmp, Papel) a decomposição da relação ProjetoRecurso(CodEmp, CodPrj, Papel).
Exemplo

- Da 4FN para a 5NF

- Explanação de que a última forma normal pode ser alcançada com projeções

Forma Normal De Boyce-Codd
Definição
Uma tabela está na BCNF se e somente se todo atributo não chave depender funcionalmente diretamente da chave primária, ou seja, não há dependências entre atributos não chave. Porem nem toda tabela que está na 3FN é uma tabela BCNF.
Exemplo

- Como transformar da 3NF para BCNF
- Nem sempre pode ser alcançada preservando a dependência.

Sexta Forma Normal ou Forma Normal Chave-Domínio
Definição
De acordo com [1]Forma Normal Chave-Domínio é uma forma atualizada na normalização de banco de dados que necessita que não haja restrições de domínio e chave dentro do banco de dados.

Esta forma[1] evita as restrições de dados que não são do domínio ou chaves com restrição. Muitos dos Banco de dados podem fazer os testes com seus domínios e chaves restritas com os seus atributos. Contudo a restrições necessitam da programação nesses bancos de dados.
Outras dependências

- dependências encapsuladas;
- dependências como blocos em lógica de primeira ordem.

Desnormalização

A criação de novas entidades e relacionamentos podem trazer prejuízos ao serem implementados em um SGBD. Devido a algumas relações excessivamente normalizadas, simples alterações no bancos de dados podem ocasionar uma profunda mudança na coerência de dados, assim entidades e relacionamentos devem ser desnormalizados para que o SGBD apresente um melhor desempenho.

Além disso, entidades podem conter ocorrências de mudanças de informações ao longo do tempo e a desnormalização pode contribuir com a manutenção de dados sem afetá-los drasticamente.
Para ilustrar o conceito podemos tomar como exemplo uma entidade denominada Vendedor com os seguintes atributos:

- Código do Vendedor;
- Nome do Vendedor;
- Região de Vendas.

Um vendedor cadastrado pode mudar sua área de vendas mas os dados de vendas antigas, realizadas em regiões por onde trabalhou, devem ser mantidos no banco sem incoerência de dados. Assim, a entidade Vendedor deve ser mantida desnormalizada para que os dados não se tornem inconsistentes.

Chave primaria	Codigo Vendedor	Nome	Regiao
001	132	Maria Auxiliadora	Natal
002	132	Maria Auxiliadora	Nova York

Nesse caso fomos obrigados a utilizar uma chave primária para manter a integridade de dados de Vendedor, o que mantém a entidade desnormalizada. Mas, ao optar pela desnomalização de dados devemos levar em conta o custo da redundância de dados e o uso das anomalias de atualização, o que não é interessante para grandes bancos de dados.

Atômicos significam que os valores não podem se repetir e nem atributos com mais de um valor. Monovalorado quer significar que os atributos possuem apenas um valor para uma entidade. Ex.: Eu não posso atribuir um CPF para mais de uma pessoa e nem atribuir o CPF e o Nome no mesmo campo.

FAILURE MODES

A gestão de banco de dados, muitas vezes, pode ser desafiadora e exige o conhecimento dos principais erros e formas de evitá-los.

A era digital tomou conta e seu negócio precisa fazer parte dessa revolução em constante mudança. Isso significa que irão precisar lidar com dados.

Entretanto, esse processo pode expor sua empresa a erros de gerenciamento e alguns riscos que consomem mais recursos.
As pessoas imaginam que as organizações precisam gastar muito dinheiro em tecnologias novas e eficientes, mas, no final, isso se resume a uma boa operação.

Sem uma forte estratégia de gestão de banco de dados, muitas empresas podem não conseguir se sustentar por muito tempo.
Assim, também deve haver um esforço para melhorar sua governança e investir em soluções e boas práticas nos processos.

Se a construção do banco de dados for feita corretamente, então o desenvolvimento, a implantação e o desempenho subsequente na produção terão menos chances de apresentar algum problema.

Os erros de entrada de dados resultam em problemas de qualidade e refletem em decisões errôneas que se mostram extremamente caras para várias funções de negócios.

Além disso, os dados imprecisos podem continuar se acumulando, especialmente quando ninguém está gerenciando.
Na prática, as organizações costumam se concentrar na coleta maciça de dados e no gerenciamento de Big Data, mas esquecem que existem algumas demandas inerentes e, ao fazê-lo, tendem a cometer alguns erros comuns.

Neste artigo, traremos uma lista com os principais erros cometidos no gerenciamento dos dados, entenderemos quais são e como evitá-los.

8 erros mais comuns

Qualquer um dos oito erros apresentados aqui impactam em custos para a sua operação que podem ser relacionados ao hardware (espaço extra em disco, largura de banda de rede).
Além disso, também podem gerar gastos relacionados ao suporte (desempenho ruim, re-design de banco de dados, criação de relatórios, etc). Confira a seguir algumas falhas relacionadas a gestão de banco de dados.

1 – Falta de uma política de backup

Antes de mais nada, ter uma política de backup definida e registrada é importante para manter o banco de dados.
Esse recurso precisa incluir procedimentos de backup de mídia, políticas de retenção e técnicas de recuperação.

Muitos modelos de negócios podem desligar um sistema em minutos ou horas com pouca perda financeira. No entanto, a perda total de dados é inaceitável, especialmente se o valor do negócio estiver nas informações.

Assim, a política de retenção determinará por quanto tempo os dados serão retidos e esse processo dependerá da área de atuação da empresa, dos gestores e do setor de TI (Tecnologia da Informação).

2 — Normalização ineficiente

A normalização é um processo que transforma uma ideia aproximada de tabelas e colunas em um design de banco de dados que segue um conjunto específico de regra, visa ser eficiente e eliminar a redundância.

Dessa forma, a gestão de banco de dados deve seguir a normalização (a menos que você esteja projetando um Data Warehouse, então você terá um conjunto diferente de regras a seguir).

De fato, esse procedimento pode variar conforme a área de atuação da empresa, a finalidade do banco de dados e o projeto do desenvolvedor.
Diferentes desenvolvedores podem aplicar as regras e usar sua experiência e compreensão dos dados e chegar a diferentes projetos.

Mas, desde que você siga as regras de normalização, seu banco de dados deve ser bem projetado.
Se não, você pode acabar com problemas com o banco de dados, como:

- Registro duplicado de informações;
- Exclusões de entradas que causam remoções de dados não intencionais em outros lugares.

3 — Produtos não licenciados

Usar software não licenciado pode ser um grande problema, já que você não terá suporte do fabricante para corrigir erros do sistema ou aplicar correções (chamadas de patches).
Algumas versões comerciais do produto oferecem opções gratuitas, mas com funcionalidade limitada.

Por isso, ao definir uma solução de banco de dados é essencial contar com um produto licenciado para ter suporte especializado sempre que precisar.

4 — Ter dados redundantes

Este é um problema semelhante à normalização, mas você ainda pode ter um banco de dados que parece normalizado, mas contém

dados redundantes (desnecessários ou que não precisam ser armazenados).
Um exemplo é a idade atual de um cliente, pois o sistema pode calcular com base em sua data de nascimento.
Outro exemplo seria o armazenamento de informações do usuário armazenadas em outros sistemas.

Por exemplo, se você usar um sistema central de gerenciamento de contas, que inclui validação de senha, então você, provavelmente, não precisa armazenar senhas em seu banco de dados.

5 — Dimensionamento mal estimado

O sistema de banco de dados requer uma gestão que faça o dimensionamento correto dos recursos usados para aperfeiçoar o banco de dados.

A instalação padrão não funcionará na maioria das situações reais e alguns itens precisam ser analisados e dimensionados adequadamente:

- O tamanho do disco;
- Número de processadores;
- Configuração da quantidade de memória usada pelo banco de dados;
- Melhorias do sistema operacional para o servidor que hospeda SGDB (Sistema de Gerenciamento de Banco de Dados);
- O número de sessões que podem ser abertas.

Se esta parte for omissa, o sistema cairá permanentemente enquanto a equipe de TI tenta todas as maneiras possíveis para determinar a causa da falha contínua.

6 – Uso dos tipos de dados errados

Quando você projeta um banco de dados, chega um ponto em que você precisa determinar quais tipos de dados cada coluna deve ter. Geralmente, existem três: número, texto e data.

Entretanto, dentro deles existem muitos outros, mas é importante fazer essa categorização corretamente.
Por exemplo, os números das contas são sempre numéricos? Se sim, seu armazenamento deve ser feito em um campo correspondente ao seu tipo. Se não, então armazene-o como texto.
Registrar dados no formato errado pode causar problemas com o armazenamento e a recuperação posterior.

7 – Versões desatualizadas

Em geral, o provedor SGDB permite que você instale a versão base para fins de teste. Mas, conforme a solução contratada, pode ser necessário aplicar um patch.
Isso resolve problemas que vão desde incompatibilidades com alguns recursos do sistema operacional até consertar o idioma do próprio banco de dados.

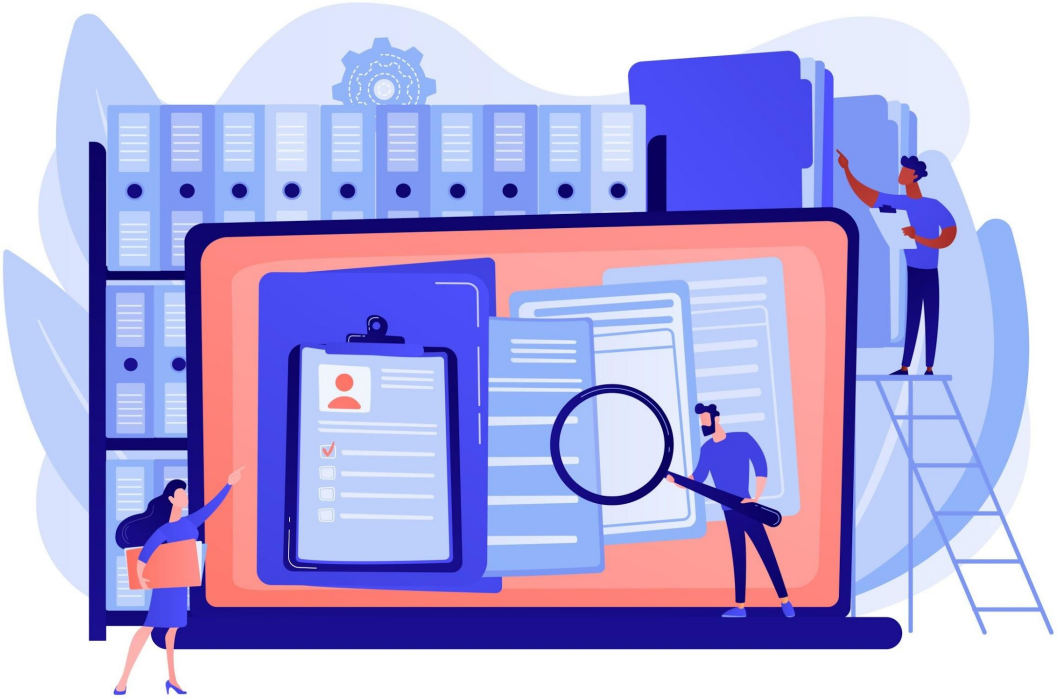
Por padrão, esses patches corrigem a instabilidade do aplicativo ou problemas de comportamento anômalo. No entanto, geralmente, o sistema será executado na versão bruta sem correções, o que acarreta falhas no sistema.
Por isso, o indicado é contar sempre com soluções que sejam completas e atualizadas pelo fornecedor.

8 – Deixar de investir em segurança

Como vimos, um servidor de banco de dados é um ativo muito estratégico e sensível.
Portanto, entendemos que o acesso só deve ser exercido por quem possua credenciais apropriadas como administradores e colaboradores que serão identificados pela política de privacidade.

Usar uma senha pessoal e segura é um requisito básico de segurança para evitar que alguém tenha a ideia de reiniciar o servidor em um horário inapropriado.
O acesso aos dados é determinado por regras de negócios e por meio de aplicativos que utilizam sua base.

Neste ponto a segurança é feita conforme os padrões da empresa devendo seguir, ainda, uma política de auditoria para as ações realizadas.
Este documento deve conter registros de conexões, endereço IP da estação, data de modificação e registro das alterações.



Dica Extra: Evite erros comuns na Entrada de Dados
As Informações inseridas da maneira errada ou ordem são mais comuns do que se imagina. Muitas vezes, as pessoas podem digitar palavras em vez de dados numéricos ou fazer o inverso.
Por isso, os erros comuns de entrada de dados são tão importantes e responsáveis pela perda de investimentos. Veja como evitar:

- Priorizando a precisão sobre a velocidade;
- Verificando todas as entradas de dados;
- Utilizando ferramentas de software para automatizar o maior número de processos;

- Treinando colaboradores sobre a importância de dados precisos;
- Proporcionando um bom ambiente de trabalho para que o ambiente inspire um trabalho com foco;
- Contratando um fornecedor de soluções para gestão de banco de dados e mão de obra interna suficiente para evitar sobrecarregar sua equipe.

Procure um especialista em gestão de banco de dados
Os erros de gestão de banco de dados podem ser difíceis de lidar, mas se você souber exatamente o que fazer, poderá evitar as falhas mais comuns.

Afinal de contas, coletar e proteger informações é uma coisa, mas você também precisa garantir que estão sendo usadas para melhorar e desenvolver os seus negócios.
Ao embarcar na gestão de banco de dados, a chave para o sucesso reside na crença de que é um processo contínuo.
A melhor abordagem é ir devagar para evitar armadilhas e atender às suas demandas organizacionais a tempo.

PROFILE PERFORMANCE

O perfil de desempenho, também conhecido como profiling, é uma técnica utilizada para analisar o desempenho de um sistema, aplicativo ou código. O objetivo é identificar áreas do código que consomem mais recursos ou que podem ser otimizadas para melhorar a eficiência. Aqui estão os principais aspectos relacionados ao profiling de desempenho:

O que é Profiling de Desempenho:

- O profiling de desempenho é o processo de medição e análise do comportamento de um sistema ou aplicativo para identificar gargalos, ineficiências e áreas de

melhoria em termos de utilização de recursos, como CPU, memória e tempo de execução.

Ferramentas de Profiling:

- Existem várias ferramentas de profiling disponíveis para diferentes linguagens de programação e ambientes. Algumas ferramentas populares incluem:
  - Para Python: cProfile, Py-Spy, line_profiler.
  - Para Java: VisualVM, YourKit Java Profiler.
  - Para C++: Valgrind, gprof, perf.
  - Para JavaScript: Chrome DevTools, Node.js Profiler.

Tipos de Profiling:

- Profiling de CPU: Identifica quais partes do código consomem mais tempo de CPU.
- Profiling de Memória: Analisa o uso da memória, identificando vazamentos ou áreas de alto consumo.
- Profiling de E/S: Foca em operações de entrada/saída para otimizar o acesso a arquivos ou bancos de dados.
- Profiling de Tempo Real: Monitora a aplicação em execução para identificar gargalos em tempo real.

Instrumentação de Código:

- O profiling geralmente envolve a adição de instrumentação ao código, inserindo pontos de medição para coletar dados sobre o desempenho durante a execução.

Análise de Resultados:

- Os resultados do profiling são analisados para identificar padrões e áreas de melhoria. Isso pode incluir a identificação de funções específicas que consomem muito tempo, alocações excessivas de memória ou operações de E/S lentas.

Otimização de Código:

- Com base nos resultados do profiling, os desenvolvedores podem realizar otimizações no código, como refatoração, uso de algoritmos mais eficientes, redução de chamadas de função desnecessárias, entre outros.

Custo de Overhead:

- A instrumentação adicionada para profiling pode introduzir um certo overhead. Portanto, é importante equilibrar a precisão da medição com o impacto no desempenho.

Profiling em Ambientes de Produção:

- Em alguns casos, é possível realizar profiling em ambientes de produção para analisar o desempenho real da aplicação em condições reais de uso. Isso deve ser feito com cuidado para evitar impactos negativos na produção.

Automação do Profiling:

- A automação do processo de profiling, incluindo a integração contínua, pode ser valiosa para identificar regressões de desempenho durante o desenvolvimento.

Uso de Ferramentas Visuais:

- Algumas ferramentas de profiling fornecem interfaces visuais que facilitam a interpretação dos resultados, permitindo uma análise mais intuitiva.

Profiling em Aplicações Web:

- Em aplicações web, o profiling pode se concentrar no desempenho do lado do servidor, bem como nas interações do lado do cliente. Ferramentas como o Chrome DevTools oferecem recursos de profiling para JavaScript.

Aprimoramento Contínuo:

- O profiling não é uma atividade única. Deve ser incorporado como parte do processo de desenvolvimento contínuo para garantir que a aplicação mantenha um desempenho ideal à medida que evolui.

O profiling de desempenho é uma prática essencial para desenvolvedores que buscam otimizar o desempenho de suas aplicações. Ao identificar e abordar efetivamente áreas de ineficiência, os desenvolvedores podem melhorar a eficiência, a escalabilidade e a experiência do usuário de seus aplicativos.

APRENDER SOBRE APIs

API - AUTENTICAÇÃO

Em um ambiente digital que exige a conexão de diferentes aplicações e sistemas, o uso de APIs para integração já é indispensável. Essa é a melhor maneira de viabilizar o funcionamento dos aplicativos simultaneamente.
Contudo, tornar esse processo seguro é um desafio. No mercado, existem muitas opções de segurança para projetar e arquitetar APIs. Além de explorar essas alternativas, também é possível diminuir as dificuldades e adotar algumas práticas recomendadas para autenticação de API.

É o que veremos neste post. Continue lendo o artigo!

3 métodos de autenticação de API

Ao usar os protocolos de autenticação padrão da indústria, você conseguirá proteger suas APIs de maneira escalável e previsível.
Dessa maneira, sua equipe pode utilizar os serviços, componentes e bibliotecas definidos, sem confundir os usuários finais.

Ao criar uma API para clientes do servidor, você terá, basicamente, três opções para implementar a autenticação. São elas:

OAuth 2.0

OAuth 2 é um protocolo de autorização que permite que uma aplicação se autentique em outra. Para que isso aconteça, uma aplicação pede permissão de acesso para um usuário, sem que para isso ela tenha acesso a alguma senha dele. O usuário pode conceder ou não o acesso à aplicação. Depois da permissão ser aceita, caso o usuário precise alterar a senha de acesso, a permissão continuará válida para a aplicação e, caso necessário, a permissão dada à aplicação pode ser revogada a qualquer momento também.

Provavelmente você já clicou em algum botão escrito “Logar com sua conta do Google” ou “Logar com sua conta do Facebook” quando você está em alguma outra aplicação, para evitar de ter que fazer na mão algum cadastro. Neste caso, você está dando a autorização de uma aplicação terceira a usar os recursos da sua aplicação, neste caso o Google ou o Facebook. Essas aplicações têm acesso limitado às informações de usuários através do protocolo HTTP. OAuth 2 é utilizado nos mais diversos tipos de autenticação, como em telas de login e na autenticação de APIs (Application Programming Interface).

Roles (papéis)

OAuth 2 foi construído em cima de 4 papéis, sendo:

- Resource Owner - pessoa ou entidade que concede o acesso aos seus dados. Também chamado de dono do recurso.
- Client - é a aplicação que interage com o Resource Owner, como por exemplo o browser, falando no caso de uma aplicação web.
- Resource Server - a API que está exposta na internet e precisa de proteção dos dados. Para conseguir acesso ao seu conteúdo é necessário um token que é emitido pelo authorization server.
- Authorization Server - responsável por autenticar o usuário e emitir os tokens de acesso. É ele que possui as informações do resource owner (o usuário), autentica e interage com o usuário após a identificação do client.

Como funciona?

Na imagem acima, podemos ver como geralmente funciona o fluxo de autorização.

1. Solicitação de autorização Nessa primeira etapa o cliente (aplicação) solicita a autorização para acessar os recursos do servidor do usuário
2. Concessão de autorização Se o usuário autorizar a solicitação, a aplicação recebe uma concessão de autorização.
3. Concessão de autorização O cliente solicita um token de acesso ao servidor de autorização (API) através da autenticação da própria identidade e da concessão de autorização.
4. Token de acesso Se a identidade da aplicação está autenticada e a concessão de autorização for válida, o servidor de autorização (API) emite um token de acesso para a aplicação. O cliente já vai ter um token de acesso para gerenciar e a autorização nessa etapa já está completa.
5. Token de acesso Quando o cliente precisar solicitar um recurso ao servidor de recursos, basta apresentar o token de acesso.
6. Recurso protegido O servidor de recursos fornece o recurso para o cliente, caso o token de acesso dele for válido.

   O servidor de autorização é responsável pelo SSO (Single Sign On), que centraliza as credenciais dos acessos dos usuários e faz a autenticação, gerencia as permissões dos usuários e emite os tokens de acesso.

O client (aplicação que roda na máquina do usuário) é definido através de dois tipos de aplicação:

- Confidential - clients que são capazes de manter a confidencialidade das suas credenciais
- Public - clients que são incapazes de manter a confidencialidade das suas credenciais

Continuando com os fluxos de autorização, existem 4 formas de se trabalhar com essa integração:

- Implicit - é um fluxo de autorização simplificado, otimizado para clients web. Ao emitir um token de acesso,

o authorization server não autentica o cliente. É muito utilizado em SPAs e aplicações MVC.

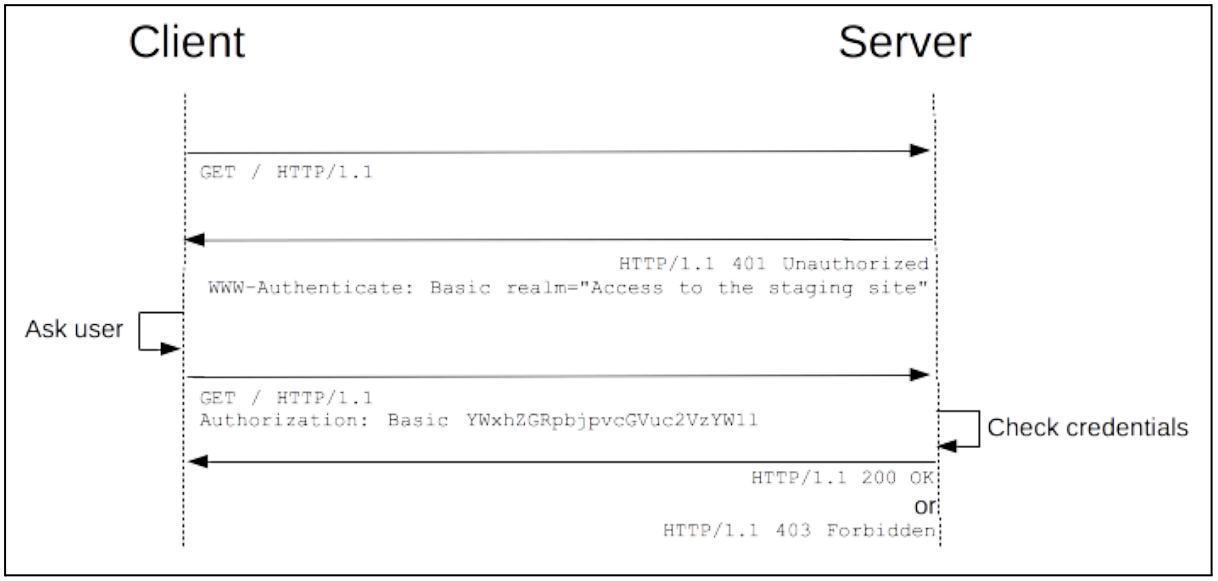
- Authorization Code - é obtido usando um authorization server como intermediário entre o client e o usuário. O cliente redireciona o usuário para um servidor de autorização. Esse tipo de client é utilizado em aplicações de terceiros, ou seja, não confiáveis.
- Resource Owner Password Credentials - utilizado quando o client solicita o usuário e senha diretamente, esse já utilizado em aplicações chamadas de confiáveis, como aplicações da própria empresa.
- Client Credentials - pode ser utilizado quando a aplicação client é protegida, que são utilizados em integrações de sistemas.
-

Autenticação básica HTTP

O HTTP fornece uma estrutura geral para controle de acesso e autenticação. A autenticação HTTP mais comum é fundamentada no esquema "Basic". Esta página introduz a estrutura HTTP para autenticação e mostra como restringir acesso ao seu servidor usando o esquema "Basic".

A estrutura geral de autenticação HTTP
RFC 7235 define a estrutura de autenticação HTTP que pode ser usada por um servidor para definir uma solicitação ("challenge (en-US)") do cliente e para um cliente fornecer informações de autenticação. A pergunta e resposta segue um caminho como esse: O servidor responde ao cliente com uma mensagem do tipo 401 (Não autorizado) e fornece informações de como autorizar com um cabeçalho de resposta WWW-Authenticate contendo ao menos uma solicitação. Um cliente que deseja autenticar-se com um servidor pode fazer isso incluindo um campo de cabeçalho de solicitação WWW-Authenticate com as credenciais. Usualmente um cliente apresentará uma solicitação de senha ao usuário e, em

seguida, emitirá uma solicitação incluindo o cabeçalho Authorization correto.



No caso de uma autorização "Basic" (como a mostrada na figura), a troca deve acontecer por meio de uma conexão HTTP (TLS) para ser segura.

Autenticação de Proxy

O mesmo mecanismo de solicitação e resposta pode ser usado para uma autenticação de proxy. Neste caso, é um proxy intermediário que requer autenticação. Como ambas autenticação de recurso e autenticação de proxy podem coexistir, um conjunto diferente de códigos de cabeçalhos e status torna-se necessário. No caso de proxys, o código de status de solicitação é 407 (Autenticação de Proxy necessária), o cabeçalho de resposta Proxy-Authenticate contém ao menos uma solicitação aplicável para o proxy, e o cabeçalho de pedido Proxy-Authorization é usado para fornecer as credenciais ao servidor proxy.

Acesso proibido

Se um servidor proxy recebe credenciais válidas, mas que não são adequadas para ter acesso a um determinado recurso, o servidor responderá com o código de status Forbidden 403. Ao contrário de 401 Unauthorized ou 407 Proxy Authentication Required, a autenticação é impossível para este usuário.

Autenticação de imagens de origem cruzada

Um potencial buraco de segurança que foi corrigido recentemente pelos navegadores é a autenticação de imagens cross-site (origem cruzada). Do Firefox 59 (en-US) em diante, recursos de imagem carregados de diferentes origens não são mais capazes de adicionar diálogos de autenticação HTTP (Erro do Firefox 1423146), impedindo que as credencias do usuário sejam roubadas se invasores conseguissem incorporar uma imagem arbitrária em uma página de terceiros.

A codificação de caracteres da autenticação HTTP

Os navegadores usam a codificação utf-8 para nomes de usuários e senhas. Firefox usava ISO-8859-1, mas alterou para utf-8 por questões de compatiblidade com outros navegadores, assim como para evitar os potenciais problemas descritos em Erro do Firefox 1419658.

Cabeçalhos WWW-Authenticate e Proxy-Authenticate

Os cabeçalhos de resposta WWW-Authenticate e Proxy-Authenticate definem o método de autenticação que deve ser usado para ganhar acesso a um recurso. Eles precisam especificar que esquema de autenticação é usado para que o cliente que deseja autorizar saiba como fornecer as credenciais. A sintaxe para esses cabeçalhos é a seguinte:

WWW-Authenticate: <type> realm=<realm>
Proxy-Authenticate: <type> realm=<realm>

<type> é o esquema de autenticação ("Basic" é o esquema mais comum e será introduzido abaixo). O realm é usado para indicar a área protegida ou o escopo de proteção. Poderia ser uma mensagem parecida com "Access to the staging site" (Acesso ao site de teste), portanto o usuário saberá qual área ele está tentando acessar.

Cabeçalhos Authorization e Proxy-Authorization

Os cabeçalhos de solicitação Authorization e Proxy-Authorization contém as credenciais para autenticar um agente de usuário com um servidor proxy. Aqui o tipo é

novamente necessário, seguido pelas credenciais, que podem ser codificadas ou criptografadas dependendo do esquema de autenticação usado.

Authorization: <type> <credentials>
Proxy-Authorization: <type> <credentials>

Esquemas de autenticação

A estrutura geral de autenticação HTTP é usado por vários esquemas de autenticação. Os esquemas podem divergir na força da segurança e na disponibilidade do software cliente ou servidor.

O esquema mais comum de autenticação é o "Basic", que é introduzido com mais detalhes abaixo. IANA mantém uma lista de esquemas de autenticação, mas existem outros esquemas oferecidos por serviços de hospedagem, como Amazon AWS. Os esquemas de autenticação comuns incluem:

- Basic (veja RFC 7617, credenciais codificadas em base64. Veja abaixo mais informações.),
- Bearer (veja RFC 6750, tokens bearer (de portador) para acessar recursos protegidos por OAuth 2.0),
- Digest (veja RFC 7616, apenas hash md5 é suportado no Firefox, veja Erro do Firefox 472823 para o suporte de encriptação SHA),
- HOBA (veja RFC 7486 (esboço), HTTP Origin-Bound Authentication (Autenticação Vinculada à Origem HTTP), baseado em assinatura digital),
- Mutual (veja draft-ietf-httpauth-mutual),
- AWS4-HMAC-SHA256 (veja Documentação AWS).

Esquema Basic de autenticação

O esquema "Basic" de autenticação HTTP é definido em RFC 7617, transmitindo credenciais como pares de ID/senhas de usuários, codificadas usando base64.

Segurança da autenticação básica

Como o ID e senha do usuário são transmitidos através da rede como texto claro (é codificado em base64, mas base64 é uma codificação reversível), o esquema básico de autenticação não

é seguro. HTTPS / TLS devem ser usados em conjunto com autenticação básica. Sem esses aprimoramentos de segurança adicionais, a autenticação básica não deve ser usada para proteger informação sensível ou valiosa.

Restringindo acesso no Apache e autorização básica

Para proteger com senha um diretório em um servidor Apache, você precisará de um arquivo .htaccess e um .htpasswd.

O arquivo .htaccess normalmente parece com isso:

```
AuthType Basic
AuthName "Access to the staging site"
AuthUserFile /path/to/.htpasswd
Require valid-user
```

O arquivo .htaccess referencia um arquivo .htpasswd em que cada linha contém um nome de usuário e senha separados por dois pontos (":"). Você não pode ver as senhas reais porque foram criptografadas (em md5, neste caso). Note que você pode renomear seu arquivo .htpasswd caso queira, mas tenha em mente que este arquivo não deve ser acessado por ninguém. (Apache normalmente é configurado para previnir acesso aos arquivos .ht*).

```
aladdin:$apr1$ZjTqBB3f$IF9gdYAGlMrs2fuINjHsz.
user2:$apr1$O04r.y2H$/vEkesPhVInBByJUkXitA/
```

Restringindo acesso no nginx e autenticação básica

No nginx, você precisará especificar uma área que que você protegerá e a diretiva auth_basic que fornece o nome para a área protegida por senha. A diretiva auth_basic_user_file aponta para um arquivo .htpasswd contentdo as credenciais do usuário criptografadas, assim como no exemplo Apache acima.

```
location /status {
    auth_basic           "Access to the staging site";
    auth_basic_user_file /etc/apache2/.htpasswd;
}
```

Acesso usando as credenciais na URL
Vários clientes também permitem que você evite o prompt de login usando uma URL codificada contendo o nome de usuário e senha como esta:
https://username:password@www.example.com/

O uso destas URLs está obsoleto. No Chrome, a parte username:password@ nas URLs é retirada por razões de segurança. No Firefox, é verificado se o site realmente Airequer autenticação e, se não, Firefox alertará o usuário com uma mensagem "Você está prestes a logar no site "www.example.com" com seu nome de usuário "username", mas o website não requer autenticação. Isso pode ser uma tentativa de enganá-lo".

Autenticação baseada em token

Um token é um dado que não tem significado ou uso por conta própria, mas que, combinado com o sistema de tokenização correto, torna-se um elemento vital para proteger sua aplicação. A autenticação baseada em token funciona garantindo que cada solicitação a um servidor seja acompanhada por um token assinado que o servidor verifica quanto à autenticidade e só então responde à solicitação.

O JSON Web Token (JWT) é um padrão aberto (RFC 7519) que define um método compacto e independente para transmitir com segurança informações entre partes codificadas como um objeto JSON. O JWT ganhou uma enorme popularidade devido ao seu tamanho compacto, que permite que os tokens sejam facilmente transmitidos por meio de strings de consulta, atributos de cabeçalho e dentro do corpo de uma solicitação POST.
Você tem interesse em começar a usar os JWTs o mais rápido possível?

Por que usar tokens?
O uso de tokens tem muitos benefícios em comparação com métodos tradicionais, como cookies.

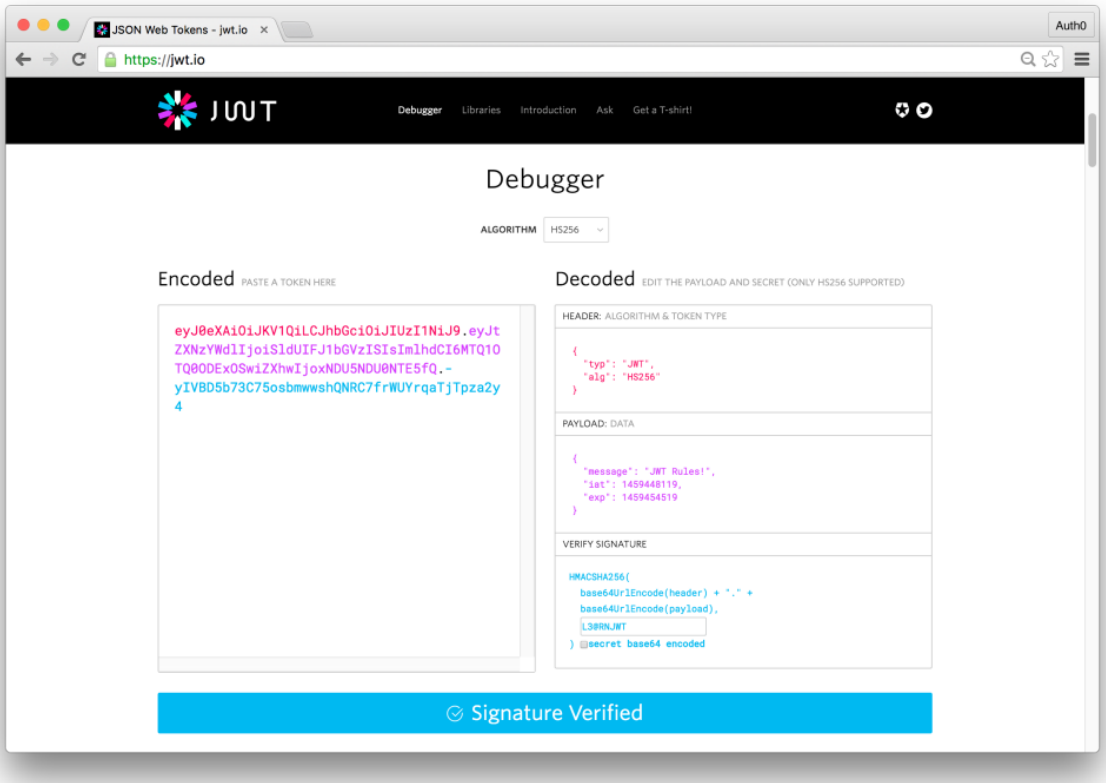
- Os tokens não tem monitoração de estado. O token é independente e contém todas as informações necessárias para a autenticação. Isso é ótimo para escalabilidade, pois libera seu servidor de armazenar o estado da sessão.
- Os tokens podem ser gerados de qualquer lugar. A geração dos tokens é dissociada da verificação, permitindo lidar com a assinatura dos tokens em um servidor separado ou até mesmo por meio de uma empresa diferente, como a Auth0.
- Controle de acesso detalhado. No payload do token, você pode especificar facilmente as funções e permissões do usuário, bem como os recursos que o usuário pode acessar.

Esses são apenas alguns dos benefícios que os JSON Web Tokens oferecem. Para saber mais, confira esta postagem do blog que se aprofunda e compara tokens a cookies para o gerenciamento de autenticação.

Anatomia de um JSON Web Token
Um JSON Web Token consiste em três partes: Cabeçalho, Payload e Assinatura. O cabeçalho e o payload são codificados em Base64 e, em seguida, concatenados por um ponto; finalmente, o resultado é assinado por algoritmo, produzindo um token na forma de header.claims.signature. O cabeçalho consiste em metadados, incluindo o tipo de token e o algoritmo de hash usado para assinar o token. O payload contém os dados das claims que o token está codificando. O resultado final se assemelha a:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJtZXNzYWdlIjoiSldUIFJ1bGVzISIsImlhdCI6MTQ1OTQ0ODExOSwiZXhwIjoxNDU5NDU0NTE5fQ.-yIVBD5b73C75osbmwwshQNRC7frWUYrqaTjTpza2y4

Os tokens são assinados para proteção contra manipulação; eles não são criptografados. O que isso significa é que um token pode ser facilmente decodificado e seu conteúdo revelado. Se

navegarmos sobre o jwt.io e colarmos o token acima, poderemos ler o cabeçalho e o payload – mas sem o segredo correto, o token é inútil e vemos a mensagem "Assinatura inválida" ("Invalid Signature"). Se adicionarmos o segredo correto, neste exemplo, a string L3@RNJWT, veremos agora uma mensagem dizendo "Assinatura verificada" ("Signature Verified").



Em um cenário real, um cliente faz uma solicitação ao servidor e passa o token com a solicitação. O servidor tenta verificar o token e, se bem-sucedido, continua a processar a solicitação. Se o servidor não puder verificar o token, envia um 401 Unauthorized (Não autorizado) e uma mensagem dizendo que a solicitação não pôde ser processada porque a autorização não pôde ser verificada.

Práticas recomendadas do JSON Web Token

Antes de começarmos a implementar o JWT, vamos abordar algumas práticas recomendadas para garantir que a autenticação baseada em token seja implementada corretamente em sua aplicação.

- Mantenha o web token em segredo. Mantenha-o em segurança. A chave de assinatura deve ser tratada como qualquer outra credencial e revelada apenas aos serviços que dela necessitam absolutamente.
- Não adicione dados confidenciais ao payload. Os tokens são assinados para proteção contra manipulação e são facilmente decodificados. Adicione o número mínimo de claims ao payload para obter melhor desempenho e segurança.
- Defina uma data de expiração para os tokens. Tecnicamente, quando um token é assinado ele é válido para sempre – a menos que a chave de assinatura seja alterada ou a expiração explicitamente definida. Isso pode acarretar problemas potenciais; portanto, tenha uma estratégia para expirar e/ou revogar tokens.
- Adote o HTTPS. Não envie tokens por conexões não HTTPS, pois essas solicitações podem ser interceptadas e os tokens comprometidos.
- Considere todos os seus casos de uso de autorização. Adicionar um sistema de verificação de token secundário que garanta que os tokens foram gerados em seu servidor, por exemplo, pode não ser uma prática comum, mas pode ser necessário para atender aos seus requisitos.

Para obter mais informações e práticas recomendadas, visite a publicação do blog “10 Things You Should Know About Tokens”(Dez coisas que você deve saber sobre tokens).

Autenticação baseada em token facilitada

A autenticação baseada em token e o JWT têm suporte amplo. JavaScript, Python, C#, Java, PHP, Ruby, Go e outros têm bibliotecas para assinar e verificar facilmente os JSON Web Tokens. Vamos implementar uma API e ver com que rapidez podemos protegê-la com JWT.

Optamos por criar nossa API com NodeJS, pois requer menos configuração. Vamos analisar o código para nossa implementação do JWT.

```
// Carregue nossas dependências
var express = require('express');
var jwt = require('jsonwebtoken');

var app = express();

// Registre a rota inicial que exibe uma mensagem de boas-vindas
// Essa rota pode ser acessada sem um token
app.get('/', function(req, res){
  res.send('Welcome to our API');
})

// Registre a rota para obter um novo token
// Em um cenário realista, autenticamos as credenciais do usuário
// antes de criar um token, mas para simplificar, acessar esta rota
// gerará um novo token válido por 2 minutos
app.get('/token', function(req, res){
          var token = jwt.sign({username:'ado'}, 'supersecret',{expiresIn: 120});
  res.send(token)
})

// Registre uma rota que requer um token válido para visualizar os dados
app.get('/api', function(req, res){
  var token = req.query.token;
```

```
  jwt.verify(token, 'supersecret', function(err, decoded){
    if(!err){
        var secrets = {'accountNumber' : '938291239','pin' :
'11289','account' : 'Finance'};
      res.json(secrets);
    } else {
      res.send(err);
    }
  })
})

// Inicie nossa aplicação na porta 3000
app.listen('3000');
```

Para testar nossa API atual, vamos executar a aplicação e navegar até **localhost:3000.** Veremos apenas a mensagem “Welcome to our API.” Em seguida, navegue até a rota **localhost:3000/api.** Uma mensagem de erro de JWT informa que não recebemos um token. Navegue até a rota **localhost:3000/token** e você verá um novo token gerado. Copie esse token; em seguida, navegue até **localhost:3000/api?token={ADD-COPIED-TOKEN-HERE}** e você verá a resposta pretendida, que são as contas financeiras da empresa.

Com apenas algumas linhas de código, conseguimos proteger nosso endpoint de API. Não abordamos o tratamento adequado da autenticação do usuário antes de gerar um token. Faremos isso com a Auth0 a seguir.

Autenticação de JWT com a Auth0

Precisamos fazer algumas pequenas modificações em nosso código para mostrar o fluxo de autenticação com a Auth0. Vamos ver as alterações abaixo:

```
// Carregue nossas dependências
var express = require('express');
var jwt = require('express-jwt');

var jwtCheck = jwt({
```

```
  secret: new Buffer('{YOUR-APP-SECRET}', 'base64'),
  audience: '{YOUR-APP-CLIENT-ID}'
});

var app = express();

// Em vez de buscar um token em nosso controlador
// usaremos um middleware, portanto, se o token for inválido,
// interromperemos a execução da solicitação
app.use('/api', jwtCheck);

app.get('/', function(req, res){
  res.send('Welcome to our API');
})

app.get('/api', function(req, res){
    var secrets = {'accountNumber' : '938291239','pin' :
'11289','account' : 'Finance'};
  res.json(secrets);
})

app.listen('3000');
```

Para testar se isso funciona, vamos iniciar o servidor e navegar até localhost:3000/api. Vemos uma mensagem dizendo que não enviamos um token de autorização. Vamos até o Auth0 Playground, adicionamos nossas credenciais e obtemos um token. Adicione o seguinte código no Playground:

```
var domain = '{YOUR-AUTH0-DOMAIN}.auth0.com';
var clientID = '{YOUR-APP-CLIENT-ID}';

var lock = new Auth0Lock(clientID, domain);
lock.show({
  focusInput: false,
  popup: true,
}, function (err, profile, token) {
  alert(token)
```

```
});
```

Para garantir que possamos obter um token, precisaremos navegar para as configurações da aplicação no Auth0 Dashboard e adicionar https://auth0.github.io/playground à nossa lista de URLs de callback. Agora vamos fazer login ou criar uma conta no Auth0 Playground e ver um pop-up que mostra nosso token.

Para verificar o conteúdo do nosso token, podemos decodificá-lo em jwt.io. Para verificar o token, precisaremos do Client Secret da aplicação Auth0 e marcar a caixa codificação base64 do segredo (secret base64 encode). Fazendo isso, agora devemos ver a mensagem “Assinatura verificada” (“Signature Verified”).

Para testar se nossa API funciona com esse token, precisamos fazer uma solicitação GET para localhost:3000/api e enviar o token em um cabeçalho de autorização. A maneira mais simples de fazer isso é usar uma aplicação como o Postman, que simplifica o teste de endpoint de API. Ao fazer a chamada, adicione um cabeçalho de autorização; para o valor, adicione Bearer {TOKEN}. Quando a chamada é feita, o middleware jwtCheck examina a solicitação, garante que o cabeçalho de autorização esteja no formato correto, extrai e verifica o token, e, se confirmado, processa o restante da solicitação. Usamos apenas as configurações padrão para mostrar os recursos do JWT, mas você pode aprender muito mais por meio de docs.

Casos de uso para a autenticação baseada em token
Vimos como é fácil implementar a autenticação JWT e proteger nossa API. Para concluir, vamos examinar os casos de uso em que a autenticação baseada em token é mais adequada.

- Aplicações de Plataforma como serviço – expor APIs RESTful que serão consumidas por uma variedade de frameworks e clientes.
- Aplicações móveis – implementar aplicações móveis nativas ou híbridas que interagem com seus serviços.

- Aplicações de página única (SPA) – criar aplicações modernas com frameworks como Angular e React.

Para obter recursos adicionais sobre como começar a usar JSON Web Tokens, confira estapublicação.

Autenticação JWT

Autenticação é um recurso indispensável em quase toda aplicação, pois devemos garantir que as informações contidas nessas aplicações só podem ser acessadas por usuários cadastrados e reconhecidos e durante o desenvolvimento de aplicações web é comum que realizemos a autenticação de usuários através de formulários de login e então guardarmos as informações de autenticação na sessão do navegador.

Mas como podemos realizar a autenticação de usuários quando estamos desenvolvendo uma API REST? Já que em uma API não temos a disposição uma sessão onde possamos guardar as informações de autenticação do usuário. Nesses casos é comum utilizarmos a Token Based Authentication e é justamente sobre esse modo de realizar a autenticação que iremos falar nesse artigo.

Para um pleno entendimento do conteudo exposto neste artigo recomendo que possua uma prévio conhecimento nos seguintes assuntos:

- **JSON:** Caso queira saber mais sobre esse tema eu recomendo a leitura do artigo O que é JSON?

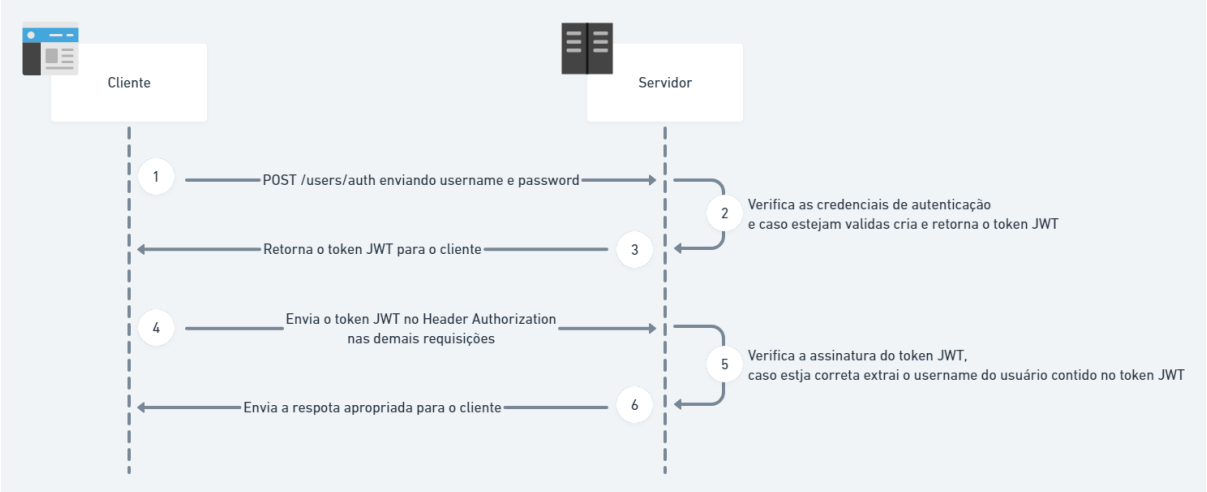
- **APIs:** Caso queira saber mais sobre esse tema eu recomendo a leitura do artigo O que é uma API?
- **Autenticação e Autorização:** Caso queira saber mais sobre esse tema eu recomendo a leitura do artigo Autenticação x Autorização
- **JWT:** Caso queira saber mais sobre esse tema eu recomendo a leitura do artigo O que é JWT?

O que é Token Based Authentication?

Token Based Authentication é uma maneira de implementar autenticação em APIs REST, onde é enviado no header Authorization da requisição um token, geralmente um token JWT, que serve como um identificador que garante que o usuário é alguém cadastrado na aplicação e já proveu suas credenciais anteriormente.

Como é o fluxo de autenticação via JWT?

Basicamente o fluxo de autenticação via JWT é composto por seis etapas.



Na imagem acima é possível observar de forma resumida um diagrama que ilustra essas seis etapas do processo de autenticação via JWT.

Agora vamos conhecer melhor cada uma dessas etapas.

Obtendo o Token JWT

A parte obtenção do token JWT engloba as três primeiras etapas de todo o processo.

Primeiro, para que o usuário seja autenticado o cliente terá que realizar uma requisição HTTP com o verbo POST para a rota de autenticação, que no nosso exemplo será a rota `/users/auth` informando no corpo da requisição as credenciais do usuário, no nosso caso os dados `username` e `password`.

Logo em seguida, a aplicação assim que recebe essa requisição terá que realizar o processo de autenticação que consiste em verificar se existe algum usuário cadastrado com o `username` informado e se o `password` informado bate com o que está salvo.

Caso as credenciais informadas no corpo da requisição estejam corretas a aplicação terá que gerar um token JWT assinado, no payload desse token devem conter as chaves `sub` que possui o identificador desse usuário, no nosso exemplo esse identificador é o `username`, também terá a chave `iat` que contém a data em que o token foi gerado em formato timestamp e por

último a chave `exp` que contém a data em que o token irá expirar também em formato timestamp.

E por fim esse token deverá ser retornado para o cliente no corpo da resposta HTTP.

Abaixo temos um exemplo da requisição.

Copiar

```
POST /users/auth HTTP/1.1
Host: localhost:8080
Content-Type: application/json
Content-Length: 52

{
  "username": "cleyson",
  "password": "senha@123"
}
```

Abaixo temos o exemplo da resposta para a requisição realizada acima.

Copiar

```
HTTP/1.1 200
Server: Tomcat
Content-Type: application/json
Date: Fri, 22 Jul 2021 11:00 GMT-3

{
                                                                "token":
"eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJjbGV5c29uIiwiZXhwIjoxNjI1ODczNTQ5LCJpY
XQiOjE2MjU4NzM1MTl9.8eYUZR2AZ0hBkX0p4-eB55CkK-I2xmeCD3udjqdzptOjrFS09zQ
CcrHRvILMJipjzve2A_KVL6T1UBgh6NmfQQ",
  "type": "Bearer",
  "expiresAt": "2021-07-22T11:30:00.000-03:00",
                                                          "refreshToken":
"eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJjbGV5c29uIiwiZXhwIjoxNjI1ODczNTc5LCJpY
XQiOjE2MjU4NzM1MTl9.rWgvsj1lJNfNaMOb30lctT_exBheio-b1K17-Fbpq6T4G8PM4cw
uLCl8AerkJVR3Ei1_3YZ5YLppOBQicglfHw"
}
```

Veja que no corpo da resposta HTTP temos um JSON que possui o token que será utilizado pelo cliente para que o mesmo possa se identificar na aplicação, a data em que esse token irá expirar, o tipo do token e o refresh token, mas iremos falar sobre o refresh token mais à frente.

Usando o Token JWT

Agora que entendemos como é realizado a etapa de obtenção do token JWT veremos como é feito o uso do token, que engloba as etapas finais de todo o processo.

Uma vez que o cliente realizou a autenticação, o mesmo tem em mãos o token JWT que é utilizado por nossa aplicação para reconhecer um usuário autenticado, então o cliente deverá enviar esse token a partir do header Authorization em todas as requisições realizadas para rotas protegidas de nossa API.

Iremos tomar como exemplo que temos a rota `/jobs` que é uma rota que pode ser acessada com o método HTTP POST para realizar o cadastro de uma vaga de emprego e essa rota é uma rota protegida.

Para que o cliente consiga ter acesso a esta rota é necessário que o mesmo ao realizar a requisição HTTP e envie o header Authorization com o token JWT no seguinte formato .

Veja abaixo um exemplo dessa requisição.

Copiar

```
POST /api/v1/jobs HTTP/1.1
Host: localhost:8080
Authorization: Bearer
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJjbGV5c29uIiwiZXhwIjoxNjI1Nzg5MTk3LCJpYX
QiOjE2MjU3ODkxNjd9.TtU-faPPJV86iTk3wEfhIIUjuCF3mwS9_4rYJ42cSl_kZaoB7SRQ
cWxEDPWvx_SEwntP8dmcMqBjN6kIsC9aMA
Content-Type: application/json
Content-Length: 228

{
  "title": "Desenvolvedor Java Sr.",
  "company": "TreinaWeb",
  "email": "contato@treinaweb.com.br",
  "techs": [
    "Java",
    "JPA",
    "Hibernate",
    "Spring Boot",
    "Spring Data JPA",
    "Spring Security"
  ],
  "status": "OPEN"
}
```

No exemplo acima o cliente fez a requisição HTTP com o método POST passando o header Authorization com o token que foi gerado anteriormente e no corpo da requisição os dados da vaga a ser cadastrada.

Agora na nossa API antes de realizar o cadastro dessa vaga a mesma deve capturar o token que foi enviado no header, verificar se esse token está assinado corretamente, verificar se o mesmo não está expirado e por fim verificar se o usuário que é identificado por esse token tem acesso a realizar tal operação.

Caso todas as verificações acima estejam corretas então o fluxo da aplicação deve seguir normalmente, o que nosso

exemplo seria cadastrar a vaga e então retornar os dados da vaga cadastrada no corpo da resposta.

Abaixo o exemplo da resposta gerada.

```
Copiar
HTTP/1.1 201
Server: Tomcat
Content-Type: application/json
Date: Fri, 22 Jul 2021 11:01 GMT-3

{
  "id": 1,
  "title": "Desenvolvedor Java Sr.",
  "description": null,
  "company": "TreinaWeb",
  "email": "contato@treinaweb.com.br",
   "techs": "Java, Hibernate, JPA, Spring Boot, Spring Data JPA, Spring Security",
  "status": "OPEN",
  "createdAt": "2021-07-22T11:01:00.340161",
  "modifiedAt": null
}
```

E o Refresh Token?

Uma vez que o token JWT estiver expirado o cliente terá que realizar uma nova requisição para a rota de autenticação enviando as credenciais do usuário, porém isso não traz uma boa usabilidade para o nosso projeto, convenhamos que ninguém gosta de ficar informando seu usuário e senha a todo momento enquanto está utilizando algum sistema.

Então, para evitar que o cliente tenha que novamente realizar uma nova autenticação sempre que o token estiver expirado podemos utilizar o refresh token.

Basicamente o refresh token é um token que tem como proposito atualizar a autenticação sem a necessidade de enviar as credenciais do usuário novamente, o refresh token pode ser implementado de várias maneiras, no nosso exemplo o refresh token também é um token JWT, porém assinado com uma chave diferente e com um tempo de expiração um pouco maior que o tempo de expiração do token de autenticação.

Quando o token de autenticação estiver expirado o cliente pode realizar uma requisição HTTP com o verbo POST para a rota de atualização da autenticação, que no nosso exemplo é a rota `/users/refresh/{refreshToken}` para gerar um novo token de autenticação.

Veja abaixo um exemplo dessa requisição.

```
Copiar
POST
/users/refresh/eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJjbGV5c29uIiwiZXhwIjoxNjI
1Nzg5OTA1LCJpYXQiOjE2MjU3ODk4NDV9.CMo2Bk4JPaeKNeNrE5qRwqPCh3Ngxnjz-RyYi
pOvH1BR83gkxSTZ5V_P6fLEwf_YA86pDEVY-DVIOGOKwYDf9A HTTP/1.1
Host: localhost:8080
Content-Length: 0
```

Uma vez que a nossa API receba essa requisição, a mesma deverá capturar o refresh token, que nesse caso foi enviado a través de uma variável na rota, verificar se o refresh token está

assinado corretamente, verificar se o mesmo não está expirado e então gerar um novo token de autenticação e um novo refresh token para o usuário referenciado pelo token.

Abaixo um exemplo da resposta para a requisição de atualização feita acima.

Copiar

```
HTTP/1.1 200
Server: Tomcat
Content-Type: application/json
Date: Fri, 22 Jul 2021 11:10 GMT-3

{
                                                                    "token":
"eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJjbGV5c29uIiwiZXhwIjoxNjI1Nzg5ODg2LCJpY
XQiOjE2MjU3ODk4NTZ9.O15AcH7AwEeb-O_J0ubpNWsO8l83JCiuDSbWBgQ8GCarvI34BR3
SFfP2SF35t0vT2p3S7KKbX-NzHkXu5zOZog",
  "type": "Bearer",
  "expiresAt": "2021-07-22T11:30:00.000-03:00",
                                                        "refreshToken":
"eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJjbGV5c29uIiwiZXhwIjoxNjI1Nzg5OTE2LCJpY
XQiOjE2MjU3ODk4NTZ9.cxyTB42lx9jr9UDChEmtrh-jukTX7X42ZIu4KmV9hjuWT2vnW6F
zr8FgwmCxBwHZddkEmdKiroy1R3tsLQgCeQ"
}
```

Dessa maneira o cliente só terá que realizar uma nova requisição para a rota de autenticação enviando as credenciais do usuário caso tanto o token de autenticação quanto o refresh token estejam expirados.

Boas práticas de autenticação de API

Para gerenciar a integração de APIs com sucesso, você precisa adotar as melhores práticas. Dessa maneira, gestão vai se tornar mais simples, segura e eficaz.

Confira, a seguir, algumas das melhores práticas de autenticação de API:

API REST

Aplicações de softwares que trabalham com interoperabilidade entre aplicações de alto desempenho precisam que essas consultas sejam realizadas de forma rápida e segura. A abstração de arquitetura de software API REST é a mais indicada na construção desses modelos de negócios.

API REST oferece um conjunto de rotinas e padrões estabelecidos para o desenvolvimento de aplicações robustas e de fácil criação, disponibilizando recursos de consumo de dados sem comprometer as regras de negócio da aplicação.

Pronto para conhecer mais sobre esse artigo? Então, confira o que falaremos a seguir:

- O que é API REST?
- Qual a relação entre HTTP e REST?
- Entenda as diferenças entre SOAP e REST
- Para que serve o API REST? Principais aplicações!
- Quais os tipos de API REST?
- O que é o Richardson Maturity Model?
- Entenda a importância do REST API!
- Quais as vantagens e desvantagens de usar API REST?

- Quais as diferenças entre API REST e API RESTFUL?
- Confira um exemplo de API rest e entenda na prática!
- Construindo uma API REST na prática: o passo a passo!
- Boas práticas para usar com API REST!

O que é API REST?

Para entendermos o que é API REST, vamos inicialmente explicitar o conceito de API.

O acrônimo API é a abreviação de *Application Programming Interface*, que significa "Interface de Programação de Aplicações". Representa um conjunto de rotinas e padrões estabelecidos e documentados para que uma determinada aplicação de software tenha autorização para utilizar as funcionalidades oferecidas por essa aplicação, sem precisar conhecer as anuências dessa implementação.

Isso garante segurança de código e, principalmente, das regras de negócio do software e a interoperabilidade entre

aplicações, em que essa comunicação ocorra utilizando as requisições HTTP responsáveis pelas operações de manipulação de dados.

API REST é uma abstração de arquitetura de software que fornece dados em um formato padronizado para modelos de requisições HTTP.

Por exemplo, sites em WordPress podem conter plugins que acessam páginas de redes sociais para tornar a interação com o conteúdo mais atrativa e interativa. Quando uma pessoa usuária clica na interação “curtir”, automaticamente uma chamada API é realizada para concluir essa ação. Isso é possível porque as redes sociais disponibilizam um token com prévia autorização para que esses sites consigam interagir de forma assíncrona.

Conheça a arquitetura REST e seus princípios!

A arquitetura REST possui um conjunto de regras e princípios que devem ser seguidos. Vamos conhecê-los:

Cliente-Servidor: Trata a respeito da separação de responsabilidades, ou seja, separar as preocupações de interface do usuário (*User Interface*) do banco de dados, abstraindo a dependência entre os lados clientes/servidor e permitindo a evolução desses componentes sem impacto e quebra de contrato.

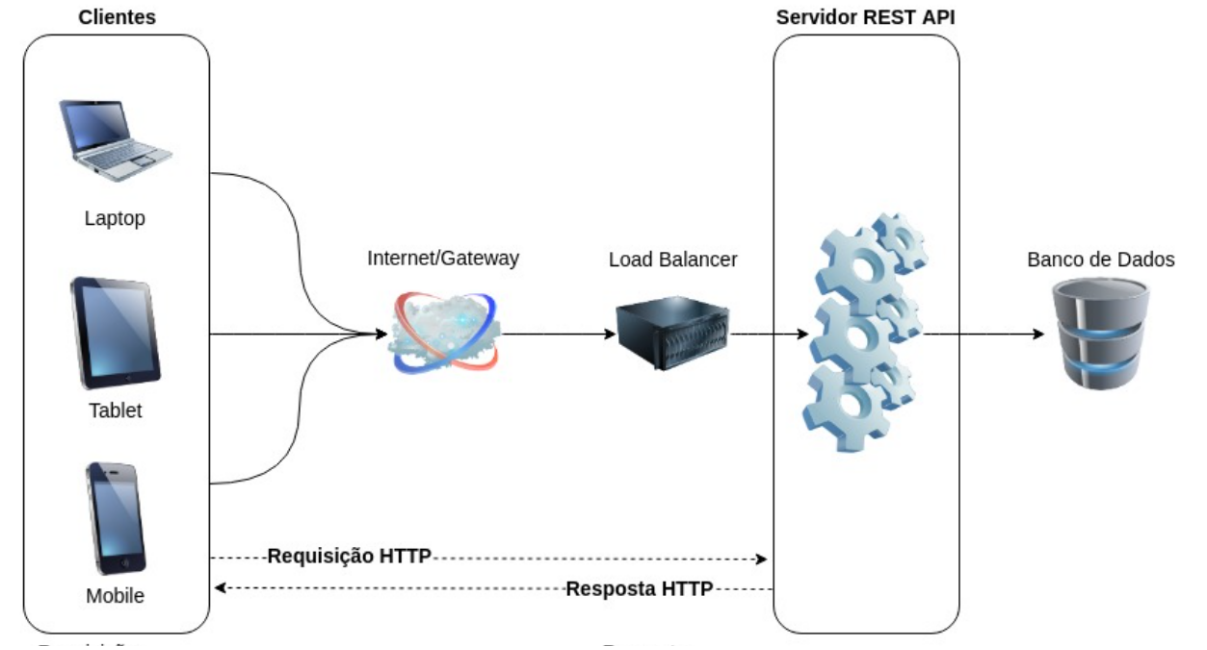
Interface Uniforme: É a interoperabilidade entre os componentes cliente e servidor. Como o cliente e servidor compartilham da mesma interface, é necessário estabelecer um contrato para a comunicação entre essas partes. Para isso, há quatro princípios a serem seguidos:
1) Identificação dos recursos (por exemplo,*Swagger*);
2) Representação dos recursos;

3) Mensagens auto-descritivas;
4) Componente HATEOAS.

Stateless: Cada requisição acionada entre a comunicação cliente-servidor deve possuir toda a informação necessária e compreensível para realizar a origem da requisição, não sendo de responsabilidade do servidor armazenar qualquer tipo de contexto. Isso pode gerar alto tráfego de dados e impacto na performance da aplicação, porém pode-se utilizar recursos de cache nesses casos.

Cache: É utilizado para melhorar a performance de comunicação entre aplicações, otimizando o tempo de resposta na comunicação entre cliente-servidor. É de responsabilidade do servidor controlar as diretivas de cache através do cabeçalho HTTP (*HTTP Header*).

Camadas: A separação de responsabilidades é importante nesse modelo de arquitetura. Os princípios e as boas práticas na arquitetura e design de um projeto, sugerem a construção de camadas independentes e auto gerenciadas, em que cada camada não pode conhecer as demais camadas. Caso ocorra mudanças em uma delas, as demais não serão impactadas. Nesse modelo, o cliente não deve conectar-se diretamente ao servidor da aplicação, porém uma camada de balanceamento de carga deverá ser acionada para essa responsabilidade.



Qual a relação entre HTTP e REST?

No estilo arquitetural REST, a manipulação dos recursos disponibilizados para o cliente é realizada através de métodos do protocolo HTTP.

Os métodos HTTP indicam os diferentes tipos de operações que o cliente pode realizar para manipular os dados através de requisições para cada contrato oferecido.

Em geral, os contratos representam as operações CRUD, porém há outros métodos HTTP que podem ser oferecidos para o cliente consumir.

É importante considerar que os métodos HTTP precisam identificar suas rotas e ações de forma clara e entendível, vejamos os exemplos a seguir:

```
POST  https://meusite.com.br/items
PUT https://meusite.com.br/items
GET https://meusite.com.br/items
```

```
POST  https://meusite.com.br/items/cadastrar
PUT    https://meusite.com.br/items/atualizar
GET https://meusite.com.br/items/listar
```

Conheça os métodos HTTP mais utilizados e seus significados:

Verbo HTTP	Descrição	Resposta
POST	É utilizado para criar um novo registro no banco de dados.	Status Code 201 – registro criado
GET	É utilizado para ler registros no banco de dados.	Status Code 200 – retorna os registros no formato solicitado.
PUT	É utilizado para atualizar um registro no banco de dados.	Status Code 200 – registro atualizado.
PATCH	É utilizado para atualizar parte de um registro no banco de dados.	Status Code 200 – registro atualizado.

DELETE	É utilizado para deletar um registro no banco de dados.	Status Code 204 – registro deletado.

Entenda as diferenças entre SOAP e REST

Enquanto no estilo arquitetural REST, a implementação dos recursos é rápida, prática e acessível, no protocolo SOAP vamos encontrar regras que desaceleram a sua utilização.

O acrônimo SOAP é a abreviação de “Simple Object Access Protocol”, significa protocolo de acesso a objetos simples, é utilizado para possibilitar a comunicação entre aplicações distintas, não importando em qual linguagem de programação elas tenham sido desenvolvidas. Para esta comunicação ocorrer, esse protocolo estabelece regras integradas que aumentam a sua complexidade e impactam no tempo de retorno dos dados.

No estilo arquitetural REST, toda a comunicação é realizada através de endpoints oferecidos para que outras aplicações, utilizando token para a segurança no acesso e manipulação dos dados, consiga interagir com segurança utilizando os métodos HTTPs disponibilizados.

No protocolo SOAP, as requisições são enviadas através de serviços SMTP (e-mail), JMS (Java Message Server), HTTP (navegadores de web), TCP (protocolo de controle de transmissão), entre outros.

Para que serve o API REST? Principais aplicações!

API REST serve para a comunicação entre aplicações para estabelecer o consumo de informações de forma rápida e segura.

É utilizada para estruturar qualquer modelo de aplicações web para os dias atuais, onde temos um alto volume de trocas de dados sendo processados de forma assíncrona entre diversas aplicações. Conheça algumas delas:

- Redes sociais
- Sites de E-Commerce
- Internet das coisas
- Inteligência Artificial
- Aplicações executadas pelo próprio navegador (Web App)
- Aplicações Mobile
- Serviços de troca de mensagens (WhatsApp, Slack)
- Repositório de dados (Google Drive, GitHub, Azure Blob Storage)

Quais os tipos de API REST?

Utilizar API REST significa utilizar uma API para o consumo de dados em aplicações back-end, de modo que essa comunicação seja estabelecida utilizando padrões definidos no estilo arquitetural REST. Os três tipos de APIs são:

Privadas ou Locais

Modelo restritivo, utilizado entre aplicações internas de uma empresa, ou seja, local.

Geralmente são aplicações que disponibilizam acessos a dados mais críticos e sigilosos da empresa, em que o acesso a aplicações externas é bloqueado utilizando Proxy e sistema de autenticação de alto nível.

Parceiras ou baseadas em programa

Modelo restritivo, utilizado entre parceiros de negócios ou para a integração entre diferentes softwares de uma mesma empresa ou entre terceiros.

Nesse modelo, a autenticação de acesso aos dados é conhecida apenas entre as partes estabelecidas.

Públicas ou baseadas em web

Modelo que pode ser acessado praticamente por qualquer aplicação de software, mediante um pedido de requisição do tipo *HTTP GET*.

Uma das suas principais utilidades é na realização de testes de acesso ou validação de uma aplicação durante o seu desenvolvimento.

Conheça alguns modelos públicos de API REST:

- Nasa
- Marvel
- Football
- Clima Tempo
- Youtube
- MovieDB

O que é o Richardson Maturity Model?

O acrônimo RMM conhecido como “Richardson Maturity Model”, significa um modelo de maturidade sugerido em 2008 por Leonard Richardson para classificar as APIs da Web, conforme sua aderência e conformidade para cada um dos quatro níveis do modelo.

Para isso, Richardson utilizou três fatores: URI, métodos HTTP e Hypermedia (HATEOAS). Um serviço é considerado maduro quando se emprega esses conceitos. Veja uma demonstração da hierarquia desses fatores:

As categorias que Richardson utilizou foram:

Level Zero Services: Nível zero de maturidade significa que uma determinada aplicação não utilizou os recursos de URI, métodos HTTP e Hypermedia (HATEOAS).

Level One Services: Nível um de maturidade considera a utilização eficiente de URIs. Os recursos são mapeados, porém não empregam com eficiência o uso dos verbos. Exemplos: métodos HTTP POST e HTTP GET.

Level Two Services: Nível dois de maturidade considera o uso eficiente de URIs e verbos HTTP.

Nesse nível, a API suporta os seguintes verbos HTTP:

– HTTP POST: utilizado para criar registro em banco de dados;

– HTTP GET: utilizado para a leitura de registros em banco de dados;

– **HTTP PUT:** utilizado para atualizar um determinado registro em banco de dados;

– **HTTP PATCH:** utilizado para atualizar parte de um determinado registro em banco de dados.

Level Three Services: Nível três de maturidade utiliza os fatores URI, HTTP e Hypermedia com eficiência. Os controles de hipermídia tem o objetivo de gerenciar o que pode ser feito, enquanto o URI do recurso ensina como manipulá-los.

Entenda a importância do REST API!

O crescente número de interações em redes sociais, o avanço da internet das coisas e a praticidade e segurança no consumo de dados entre distintas aplicações são os principais fatores que justificam a utilização do REST API para facilitar a comunicação entre esses sistemas.

REST API permite a criação de websites e aplicações em nuvem com maior rapidez e robustez, permitindo que os dados sejam acessados, criados e atualizados sincronizadamente, sem comprometer a integridade das informações.

Outro fator importante é a melhora significativa na qualidade e maturidade de software, pois quanto mais as aplicações interagem entre si para as trocas de informações, melhor será o resultado para as pessoas usuárias finais em suas tomadas de decisões.

Quais as vantagens e desvantagens de usar API REST?

API REST apresenta vantagens competitivas, porém também há desvantagens que precisam ser consideradas. São elas:

Vantagens

· Separação entre cliente-servidor

· Praticidade no acesso aos contratos da aplicação

· Confiabilidade e segurança na aplicação

· Escalabilidade para aplicações, principalmente nos modelos de microsserviços

· Multiplataforma, considerando que os dados podem retornar nos padrões XML e JSON

Desvantagens

· Conhecimento dos padrões de projeto para o modelo REST API

· Implementação “Lo-rest”, ou seja, desenvolver aplicações que disponibilizam apenas dois endpoints: HTTP GET e HTTP POST

· Trabalham de forma assíncrona, aumentando a sobrecarga das solicitações, podendo comprometer o desempenho da comunicação entre as aplicações

Quais as diferenças entre API REST e API RESTFUL?

API REST e API RESTFUL possuem objetivos distintos, mas complementares. A diferença encontra-se somente no cumprimento das exigências da arquitetura de software.

O estilo de arquitetura de uma API REST deve possuir alguns princípios:

· Não possui criptografia

· Não possui sessão

· Utiliza apenas o protocolo HTTP

· Os recursos devem ser acessados somente utilizando o protocolo de internet URI (Uniform Resource Identifier)

· O retorno dos dados devem ser nas formas mais conhecidas, exemplos: JSON e XML

API RESTFUL é o cumprimento de todos os princípios citados anteriormente, ou seja, um serviço baseado em REST é chamado de serviços RESTFUL, com o objetivo de oferecer uma excelente interatividade e rapidez na comunicação entre os serviços.

Confira um exemplo de API rest e entenda na prática!

Apresentamos o exemplo de consumo de uma API REST, onde a URI será “https://meusite.com.br”.

Os dados consultados pertencem à tabela Pessoa e seus atributos são: Id, Nome e Idade.

Vamos utilizar como retorno dos dados o padrão de formatação JSON.

1) HTTP GET

URI: https://meusite.com.br/api/pessoas

Definição: Retornar todos os dados cadastrados na tabela pessoa

Retorno dos dados:


```
{
   "status":200,
   "data":[
   {
      "id":1,
      "nome":"Maria Santos",
      "idade":20
   },
   {
      "id":2,
      "nome":"Sandro Pereira",
      "idade":25
   }
   ]
}
```

2)HTTP GET

URI: https://meusite.com.br/api/pessoas/{id}

Definição: Retornar os dados cadastrados na tabela pessoa apenas do *id* informado

Retorno dos dados:


```
{
   "status":200,
   "data":[
   {
     "id":1,
     "nome":"Maria Santos",
     "idade":20
   }
   ]

}
```


3)HTTP POST

URI: https://meusite.com.br/api/pessoas

Definição: Criar um novo registro de dados na tabela pessoa. Os campos obrigatórios devem ser informados no corpo da requisição

Corpo da requisição:

```
{
     "nome":"Joana Marques",
     "idade":32
}
```

Retorno dos dados:

```
{
   "status": 201,
   "success": true

}
```

4) HTTP PUT

URI: https://meusite.com.br/api/pessoas
Definição: Atualizar um registro de dados na tabela pessoa. Os campos a serem
atualizados devem ser informados no corpo da requisição

Corpo da requisição:

```
{
  "id":1,
     "nome":"Maria Soares dos Santos",
     "idade":21
}
```

Retorno dos dados:

```
{
   "status":200,
   "success": true

}
```

5)HTTP PATCH

URI: https://meusite.com.br/api/pessoas/{id}

Definição: Atualizar apenas determinados campos de um registro na tabela pessoa.

Corpo da requisição:

```
{
     "idade":23
}
```

Retorno dos dados:

```
{
   "status":200,
   "success": true
}
```

6) HTTP DELETE

URI: https://meusite.com.br/api/pessoas/{id}

Definição: Remover um registro na tabela pessoa.

Retorno dos dados:

```
      {
          "status":204
       }
```

Construindo uma API REST na prática: o passo a passo!

Nesse tutorial, vamos aprender a criar uma API REST utilizando a linguagem de programação C#, com base no framework gratuito .NET CORE.

Pré-Requisitos

- Editor de código: vamos utilizar a ferramenta gratuita de editor de código Visual Studio Community. O download pode ser realizado no portal da Microsoft.
- POSTMAN: os testes dos endpoints dessa aplicação serão realizados usando a ferramenta gratuita API Client Postman. O download pode ser realizado no site oficial Postman.com.

Boas práticas para usar com API REST

Conheça as boas práticas na utilização de API REST:

Organização da semântica dos serviços: uma boa semântica é quando esses serviços são de fácil leitura e compreensão. Exemplos:

Boas práticas	Evitar
[GET] https://meusite.com.br/api/peoples	[GET] https://meusite.com.br/api/getAllPeoples
[GET] https://meusite.com.br/api/peoples/1	[GET] https://meusite.com.br/api/getPeopleById

[POST] https://meusite.com.br/api/peoples	[POST] https://meusite.com.br/api/CreatePeople

Verbos HTTP: utilizar corretamente os verbos HTTP, de acordo com suas definições e operações. Os verbos mais utilizados são:

– GET: Recupera informações de um recurso;

– POST: Cria um novo recurso;

– PUT: Atualiza um determinado recurso;

– DELETE: Remove um determinado recurso;

– PATCH: Atualiza parte de um determinado recurso.

HTTP Status Code: é o padrão utilizado para as respostas de cada solicitação de serviço utilizado. Para cada verbo HTTP, os retornos do status code seguem os seguintes padrões:

– *HTTP GET:* os retornos esperados são:

  – 200 (OK): em caso de sucesso.

  – 404 (not found): caso a entidade solicitada não seja encontrada.

– *HTTP POST*: os retornos esperados são:

  – 201 (created): um novo recurso foi criado com sucesso.

– 400 (bad request): a requisição contém dados inválidos.

– 422 (unprocessable entity): a requisição violou alguma regra de negócio da aplicação.

– *HTTP PUT*: os retornos esperados são:

– 200 (OK): atualizado com sucesso.

– 400 (bad request): a requisição contém dados inválidos.

– 409 (conflit): não foi possível atualizar um recurso existente.

– *HTTP DELETE*: os retornos esperados são:

– 204 (not content): sucesso na remoção do recurso.

– 404 (not found): caso a entidade solicitada não seja encontrada.

Versionamento de API: permite maior controle na implementação dos endpoints, permitindo direcionar os clientes que utilizam esses serviços para os novos endpoints sem perda do contrato. Exemplos:

– URI: https://meusite.com.br/api/v1/pessoas

– Subdomínio: https://v1.meusite.com.br/api/pessoas

– Header: “Accept”=“application/meusite.api.v1.json”

– Querystring: https://meusite.com.br/api/pessoas?version=1.0

API REST é um estilo arquitetural que dinamiza as aplicações que necessitam trabalhar com interoperabilidade com outras aplicações na troca de dados, além de oferecer segurança e praticidade na sua implementação e manutenção.

API - gRPC

O que é gRPC?

Antes de falar sobre gRPC, precisamos saber o que é o RPC. Para não misturar os assuntos e conceitos, RPC é a definição de um protocolo para executar procedimentos em outros computadores em rede, e não cabe ao RPC especificar como a mensagem é enviada de um processo para o outro.

O gRPC é um framework do Google que implementa RPC, e tem inúmeras vantagens para a escolha do gRPC.

Principais vantagens

- Desenvolvimento de API Contract-first, com Protocol Buffer por padrão;
- Disponível em mais de 10 linguagens de programação;
- Suporte a chamadas streaming do cliente para o servidor, do servidor para cliente e bidirecional (falarei mais deste assunto);
- Reduz a utilização da rede, latência, pois os dados são trafegados em binário;
- Utiliza o protocolo HTTP2.

Protocol Buffer

Protocol buffer ou protobuf é um método criado pelo Google de serialização de dados estruturados, agnóstico de linguagem. A transferência de dados chega a ser até 6x mais rápida que um

JSON. O gRPC utiliza o arquivo com extensão .proto para criar o código base, garantindo o Contract-first.

A serialização/deserialização faz um uso menos intensivo da CPU pelo fato das mensagens estarem em formato binário, ou seja, mais próximo de como o computador representa os dados.

Cliente e o servidor em qualquer linguagem

Atualmente, o Google tem disponibilizado o gRPC em mais de 10 linguagens. Podendo utilizar qualquer uma no cliente e no servidor.

Principais vantagens do HTTP2

Citarei as principais vantagens, mas não entrarei em detalhes, já existe bastante conteúdo na internet falando sobre isso.

- Fluxo multiplexados
- Compressão do cabeçalho
- Protocolo binário

Demonstração de como o http2 é mais rápido que o http1.1. Se você tiver uma internet muito rápida, pode experimentar colocar no navegador no modo low-end mobile

Mão na massa

Para este tutorial, farei um crud de usuários no mongodb, vou utilizar o servidor e o cliente no mesmo projeto em java.
Dependências que estou utilizando no servidor. Para outras configurações, vou deixar o link do projeto no github.

```
dependencies {
   testCompile group: 'junit', name: 'junit', version: '4.12'
   implementation 'io.grpc:grpc-netty-shaded:1.33.1'
   implementation 'io.grpc:grpc-protobuf:1.33.1'
```

```
   implementation 'io.grpc:grpc-stub:1.33.1'
       implementation  "io.grpc:grpc-services:1.33.1"       //
reflection

   testCompile group: 'junit', name: 'junit', version: '4.12'

    compile group: 'org.mongodb', name: 'mongodb-driver-sync',
version: '4.1.1'
}
```

SERVER

1. Precisamos definir o .proto que será o contrato da nossa API. Arquivo user.proto. Observe que a operação ListUser, o servidor irá mandar os usuários por stream na mesma conexão tcp, e o cliente irá ficar recebendo os usuários em tempo real. Para as operações CreateUser e DeleteUser será unário, para cada request, uma response.

Não vou fazer stream do cliente para o server e nem bidirecional, pois o artigo ficará gigante, mas a forma como fazemos não é muito diferente.

```
syntax = "proto3";

package user;

option java_package = "com.proto.user";
option java_multiple_files = true;

message User {
   string id = 1;
   string name = 2;
   string email = 3;
}

message CreateUserRequest {
   User user = 1;
}

message CreateUserResponse {
```

```
   User user = 1;
}

message DeleteUserRequest {
   string userId = 1;
}

message DeleteUserResponse {
   string userId = 1;
}

message ListUserRequest {

}

message ListUserResponse {
   User user = 1;
}

service UserService {
              rpc     CreateUser(CreateUserRequest)     returns
(CreateUserResponse) {};
              rpc     DeleteUser(DeleteUserRequest)     returns
(DeleteUserResponse) {}; // return NOT_FOUND if not found
          rpc   ListUser(ListUserRequest)   returns   (stream
ListUserResponse) {};
}
```

2. Agora precisamos executar o plugin que gerará todo o código que o gRPC irá usar com base no que definimos no user.proto. O build do projeto faz este trabalho automaticamente. Não precisamos nos preocupar como os arquivos foram gerados, apenas com a implementação deles.
3. Feito isto, precisamos subir nosso servidor para receber requests, mas ainda não temos nenhum serviço implementado.

```
package br.com.crudgrpc.server;

import io.grpc.Server;
import io.grpc.ServerBuilder;
import java.io.IOException;

public class UserServer {
    public static void main(String[] args) throws IOException,
InterruptedException {
        System.out.println("Server start");
        Server server = ServerBuilder.forPort(50051)
                .build();
        server.start();
        Runtime.getRuntime().addShutdownHook(new Thread(() -> {
            server.shutdown();
        }));
        server.awaitTermination();
    }
}
```

4. Como definimos três operações no serviço UserService, iremos implementar apenas a lógica de como cada uma irá proceder, o gRPC irá criar tudo que precisamos para implementar.

Para isto, criei o arquivo UserServiceImpl com algumas configurações para conectar no mongodb e as operações. A primeira operação será a CreateUser, é uma request unária, ou seja, uma request, uma response.

```
@Override
public void createUser(CreateUserRequest request,
StreamObserver<CreateUserResponse> responseObserver) {
    System.out.println("Creating User");
    User user = request.getUser();
    Document doc = new Document("name", user.getName())
            .append("email", user.getEmail());
    collection.insertOne(doc);
```

```
   String userID = doc.getObjectId("_id").toString();
   System.out.println("Inserted user: " + userID);
   CreateUserResponse response =
CreateUserResponse.newBuilder()
           .setUser(user.toBuilder().setId(userID).build())
           .build();
   responseObserver.onNext(response);
   responseObserver.onCompleted();
}
```

5. Operação deletar usuário: esta operação tem um tratamento para usuário não encontrado, assim como fazemos nas rest apis.

```
@Override
public void deleteUser(DeleteUserRequest request,
StreamObserver<DeleteUserResponse> responseObserver) {
   String userID = request.getUserId();
   DeleteResult result = null;
   try{
       result = collection.deleteOne(eq("_id", new
ObjectId(userID)));
   }catch (Exception e){
       System.out.println("User not found");
       responseObserver.onError(
               Status.NOT_FOUND
               .withDescription("User not found for id: " +
userID)
               .augmentDescription(e.getLocalizedMessage())
               .asRuntimeException()
       );
   }
   if(result.getDeletedCount() == 0){
       System.out.println("User not found for id: " + userID);
       responseObserver.onError(
               Status.NOT_FOUND
                       .withDescription("User not found for
id: " + userID)
                       .asRuntimeException()
```

```
        );
    }else{
        System.out.println("User was deleted");
        responseObserver.onNext(
                DeleteUserResponse.newBuilder()
                .setUserId(userID)
                .build()
        );
        responseObserver.onCompleted();
    }
}
```

6. Operação listar usuários: nesta operação, o cliente fica com uma conexão aberta recebendo os usuários por stream.

```
@Override
public void listUser(ListUserRequest request,
StreamObserver<ListUserResponse> responseObserver) {
   System.out.println("Streaming users");
   collection.find().iterator().forEachRemaining(document ->
responseObserver.onNext(
           ListUserResponse.newBuilder()
           .setUser(this.documentToUser(document))
           .build()
   ));
   responseObserver.onCompleted();
}

private User documentToUser(Document document){
   return User.newBuilder()
           .setName(document.getString("name"))
           .setEmail(document.getString("email"))
           .build();
}
```

7. Para finalizar o servidor, precisamos disponibilizar o serviço que implementamos para ele. Apenas adicionei a linha 11, abaixo da instância do server.

```
package br.com.crudgrpc.server;

import io.grpc.Server;
import io.grpc.ServerBuilder;
import java.io.IOException;

public class UserServer {
    public static void main(String[] args) throws IOException,
InterruptedException {
       System.out.println("Server start");
       Server server = ServerBuilder.forPort(50051)
               .addService(new UserServiceImpl())
               .build();
       server.start();
       Runtime.getRuntime().addShutdownHook(new Thread(() -> {
           server.shutdown();
       }));
       server.awaitTermination();
   }
}
```

Client

Para o cliente, vou implementar as operações de criar, deletar e listar usuário.

1. Precisamos criar um channel para comunicar com o servidor.

```
ManagedChannel channel =
ManagedChannelBuilder.forAddress("localhost", 50051)
       .usePlaintext()
       .build();
UserServiceGrpc.UserServiceBlockingStub userClient =
UserServiceGrpc.newBlockingStub(channel);
```

2. Criar novo usuário.

```
User user = User.newBuilder()
       .setName("luiz")
```

```
        .setEmail("teste@teste.com")
        .build();
CreateUserResponse createUserResponse = userClient.createUser(
        CreateUserRequest.newBuilder()
                .setUser(user)
                .build()
);

System.out.println(createUserResponse.toString());
```

3. Deletar usuário.

```
String userId = createUserResponse.getUser().getId();
DeleteUserResponse deleteUserResponse = userClient.deleteUser(

DeleteUserRequest.newBuilder().setUserId(userId).build()
);
System.out.println(deleteUserResponse.toString());
```

4. Listar usuários. Perceba que aqui estamos usando streaming para enviar os usuários do servidor para o cliente na mesma conexão TCP.

```
userClient.listUser(ListUserRequest.newBuilder().build()).forE
achRemaining(
        listUserResponse ->
System.out.println(listUserResponse.getUser().toString())
);
```

GRAPHQL

O QUE É GRAPHQL?

GraphQL é uma query language para APIs, ou seja, é uma linguagem de consulta de dados em APIs, que foi desenvolvida pelo Facebook. O facebook utiliza o GraphQL em suas aplicações desde 2012 e somente em junho de 2015 eles liberaram.

É uma alternativa mais eficiente, poderosa e flexível em relação ao REST. As consultas são interpretadas em tempo de execução no servidor usando um sistema de tipos que você define para seus dados.

Ao trabalhar com GraphQL você irá perceber que ele é uma abstração do protocolo HTTP. Mas como assim? Basicamente, na aplicação que estiver subindo o server GraphQL, você estará utilizando um único endpoint. Este server irá receber request do tipo POST e GET (sendo POST o mais utilizado), tendo como resposta um formato JSON. Parece estranho para quem está acostumado com REST, em que temos vários endpoint, mas aguarde um pouco que você irá entender a ideia.

Uma pequena confusão com o Graphql

É muito comum a princípio as pessoas acharem que GraphQL é um banco de dados. Mas definitivamente, NÃO É. Veja bem, o fato de ter Query Language no nome não implica que ele é banco de dados, mas sim que é uma Linguagem de Consulta para APIs. Em resumo, o mesmo não está atrelado a nenhum banco de dados ou qualquer sistema de armazenamento, não é ORM. Na verdade, ele nem precisa de banco de dados para funcionar.

Fechando este tópico sobre o que é GraphQL, algumas de suas características seriam:

- Fornece a liberdade ao cliente de especificamente quais dados irá precisar.
- Trabalha com um sistema de tipos, sendo este, responsável por definir os dados.
- Fazendo uma analogia, é como se fosse uma camada ali no nosso backend que tem a capacidade de buscar dados de outras fontes (outros bancos de dados, outras APIs por exemplo).

ENTENDENDO OS CONCEITOS BÁSICOS EM TORNO DO GRAPHQL

Type System

Sistema de tipos. Tudo no GraphQL é tipado, sendo que o mesmo tem seu próprio sistema de tipos que nos permite descrever os dados da nossa API. Podemos ver um exemplo disso na Figura 1, onde foi criado um projeto no VSCode e um arquivo chamado "schema.graphql".

Figura 1: Category and Subcategory Type

```
type Category {
    uuid: ID!
    description: String
    subcategories: [Subcategory]
    status: Boolean
}

type Subcategory {
    uuid: ID!
    description: String
    status: Boolean
}
```

Observe duas particularidades: a primeira é o sinal de "!" no ID. Isto significa que não é permitido aceitar null como valor. O segundo ponto é o campo "subcategories". Podemos assumir ali que há um relacionamento entre Category e Subcategory, sendo que Category podem ter N subcategories. Mais detalhes dos tipos veremos um pouco mais adiante.

Schema

E o que contém todos os Types da nossa API. Em síntese, seu papel é definir o esquema da API. Ela engloba nossas Queries, Mutations, Subscriptions, Diretivas, etc. Exemplo:

```
type Schema {
query: Query
mutation: Mutation
subscription: Subscription
}
```

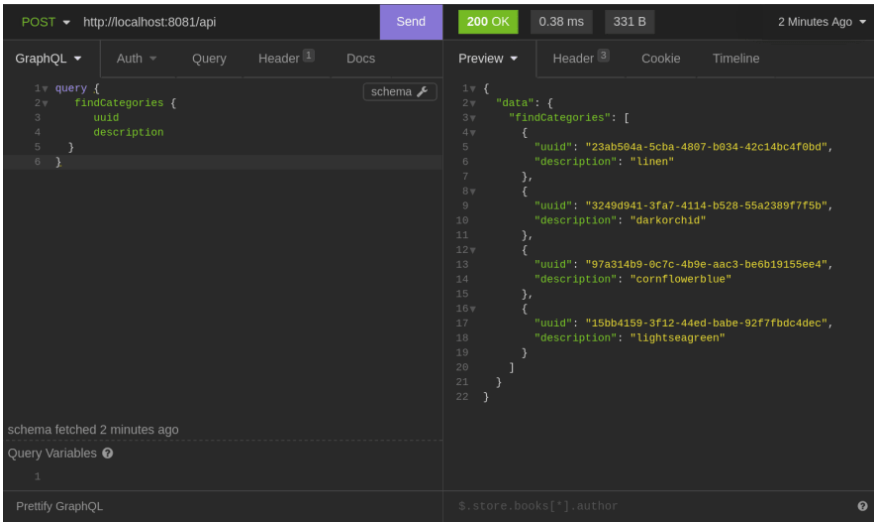
Query

Quando você quer realizar uma consulta, uma busca dos dados na API. A Figura 2 mostra a declaração de uma query em nosso

arquivo “schema.graphql”. A Figura 3 mostra uma requisição feita pelo Insomnia a nossa API e seu retorno em JSON. Detalhes da implementação da API ficará para outro momento:

```
type Category {
    uuid: ID!
    description: String
    subcategories: [Subcategory]
    status: Boolean
}

type Subcategory {
    uuid: ID!
    description: String
    status: Boolean
}

type Query {
    findCategories: [Category]
}
```

Figura 3: Realizando uma query à nossa API



Contextualizando: eu criei uma API GraphQL simples, escrita em Golang. Nesta API, temos a implementação para devolvermos estes dados. Por fim, utilizei o Insomnia para realizar a request a minha API, puxando todas as categorias cadastradas.

Temos algumas observações aqui: em nosso schema, repare que informamos o type Query. Este tipo costuma ser chamado de RootQuery. Poderíamos comparar que o RootQuery será a

instância Pai de todas as queries que podemos criar. Neste nosso exemplo, temos apenas findCategories.

Um outro ponto muito importante sobre as queries: os campos são resolvidos paralelamente. Comentarei um pouco disso abaixo.

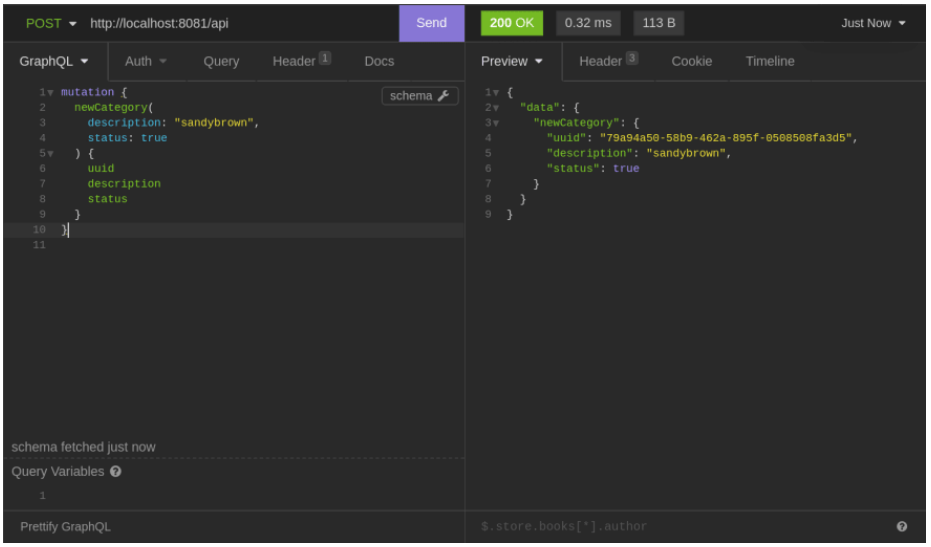
Mutation

Será o responsável por fazer mudanças dos dados na API. A ideia é que as Mutations nos permite criar, alterar e excluir dados. A Figura 4 mostra a declaração de uma Mutation. A Figura 5 mostra uma mutation sendo requisitada a nossa API pelo Insomnia:

Figura 4: Mutation

```
type Category {
    uuid: ID!
    description: String
    subcategories: [Subcategory]
    status: Boolean
}

type Subcategory {
    uuid: ID!
    description: String
    status: Boolean
}

type Query {
    findCategories: [Category!]!
}

type Mutation {
    newCategory(description: String, status: Boolean): Category
}
```

Figura 5: Realizando uma mutation à nossa API



De maneira análoga a Query, em nosso schema, repare que informamos o type Mutation. Este tipo costuma ser chamado de RootMutation. Poderíamos comparar que o RootMutation será a instância Pai de todas as mutations que podemos criar. Neste nosso exemplo, temos apenas newCategory.

Já nas mutations, diferentemente das queries, os campos são resolvidos em série, um após o outro.

Mas então, dissemos anteriormente que na query os campos são resolvidos paralelamente e as mutations em série. O que isto quer dizer? Pois bem, isto é de extrema relevância pelo fato de que se você quiser escrever dados em sua API utilizando query ao invés de mutation isto é possível. No entanto, assim como muita coisa no desenvolvimento de software, aqui no GraphQL também temos um padrão a seguir. Pelo fato de que mutations são resolvidos em série, há o risco de que se você for utilizar query para realizar alterações na API, isto pode causar um efeito indesejado, pelo fato de que queries são resolvidas paralelamente. Devido a isto, siga sempre este padrão: query para ler dados, mutation para escrever dados.

Subscriptions

Você utiliza subscription quando você precisa trabalhar em Real Time, escutando por mudanças na API. É meio que você precisará de um Web Socket para fazer isto.

Resolvers

Esta parte é muito importante de entender. Repare que apontamos os nossos tipos, declaramos nossas queries e mutations por exemplo. No entanto, apenas com esta declaração sua API não está pronta. Não como se fosse uma coisa mágica, por exemplo, que a query findCategories irá no banco de dados sem que você implemente algo para realizar este comportamento. Para isto temos “Resolver”.

É importante entender que cada campo no GraphQL possui uma função Resolver. Por exemplo, a partir da query findCategories poderia ser gerada uma função resolver para que, se o cliente chamasse esta query, sua API devolveria uma resposta, o json de categorias neste caso. Este comportamento é feito em uma função Resolver.

A Figura 6 mostra um exemplo disso, em que irei abordar mais detalhadamento em um próximo artigo (quando construíremos nossa API GraphQL utilizando Golang):
Figura 6: Função resolver para query “findCategories”

```
func (r *queryResolver) FindCategories(ctx context.Context) ([]model.Category, error) {
    categories := []model.Category{
        {
            UUID:        "23ab504a-5cba-4807-b034-42c14bc4f0bd",
            Description: "linen",
        },
        {
            UUID:        "3249d941-3fa7-4114-b528-55a2389f7f5b",
            Description: "darkorchid",
        },
        {
            UUID:        "97a314b9-0c7c-4b9e-aac3-be6b19155ee4",
            Description: "cornflowerblue",
        },
        {
            UUID:        "15bb4159-3f12-44ed-babe-92f7fbdc4dec",
            Description: "lightseagreen",
        },
    }

    return categories, nil
```

Repare que nesta função você poderia estar buscando as categorias em um banco de dados por exemplo. Apesar de não termos mostrado na Figura 6, uma função resolver pode receber 4 argumentos: parent (ou object), args, context e info.

parent: é como se fosse o "pai" do resolver naquele momento. Por exemplo, na Figura 2, se chamássemos o campo "subcategories" na query, o root para este resolver seria "findCategories".

args: seriam os argumentos que você enviasse. Se você quisesse, por exemplo, buscar uma categoria pelo UUID, o argumento "args" teria este valor para que você pudesse realizar a consulta.

context: aqui você pode fornecer valores do contexto, como o usuário conectado no momento, uma instância do banco de dados, dentre outras coisas.

info: seriam as informações específicas para a query ou mutation que está sendo invocada, bem como detalhes do schema. Exemplo, você poderia saber aqui quais campos foram requisitados na query naquele momento.

Trivial Resolvers

Aqui também é muito interessante porque será o momento em que você resolverá os campos daquele type. Como assim? Ainda na Figura 2, se precisássemos mostrar todas as subcategorias associado as categorias? Se pensarmos em banco de dados, "subcategories" seria outra tabela associada a "categories". Portanto, ao invocar este campo na query, você poderia criar uma função resolver que resolverá apenas este campo. Esta função é considerada um Resolver Trivial.

Scalar Types

Até agora, vimos que um objeto GraphQL tem um nome e seus campos. No entanto, estes campos precisam ser resolvidos de maneira concreta. Ou seja, precisamos dos Tipos Escalares. Basicamente, os tipos escalares são:

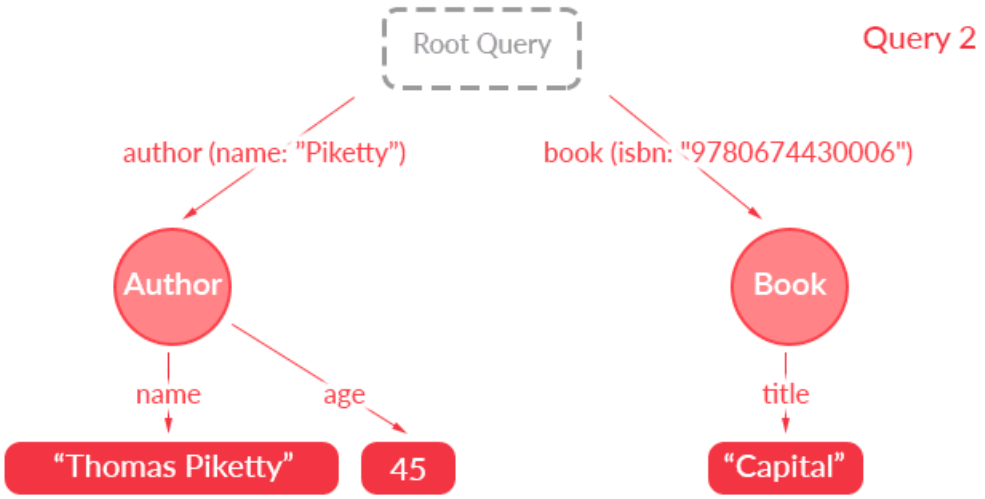
- Int: um inteiro de 32 bits (assinado)
- Float: ponto flutuante de dupla precisão (assinado)
- String: Sequência de caracteres
- Boolean: true ou false
- ID: Representa um identificador único.

É possível também trabalhar com Custom Scalar Types. Um exemplo e o que é bem comum de se ver seria um tipo data. Sintaticamente seria assim:
scalar Date
Após isto, é necessário implementar uma forma serializar ou deserializar este tipo.

CARACTERÍSTICAS DO GRAPHQL E UMA PEQUENA COMPARAÇÃO COM O REST

GraphQL é uma árvore

Observando as figuras anteriores em que trabalhamos com GraphQL, esta maneira em que os campos são resolvidos no GraphQL seria similar a uma estrutura de dados bem conhecida, árvore. Isto pode ser visualizado na Figura 7:
Figura 7: GraphQL is a tree



GraphQL é melhor que REST

Há algumas discussões em torno disso e sabemos que tudo na TI depende. No entanto, eu comungo desta opinião. Com o passar dos anos, APIs REST se mostraram inflexíveis em alguns contextos. O GraphQL foi desenvolvido com isto em mente, para lidar com estas necessidades com maior eficiência e flexibilidade. Se você já trabalhou com REST, alguns pontos abaixo você vai entender melhor ainda o que o GraphQL resolve:

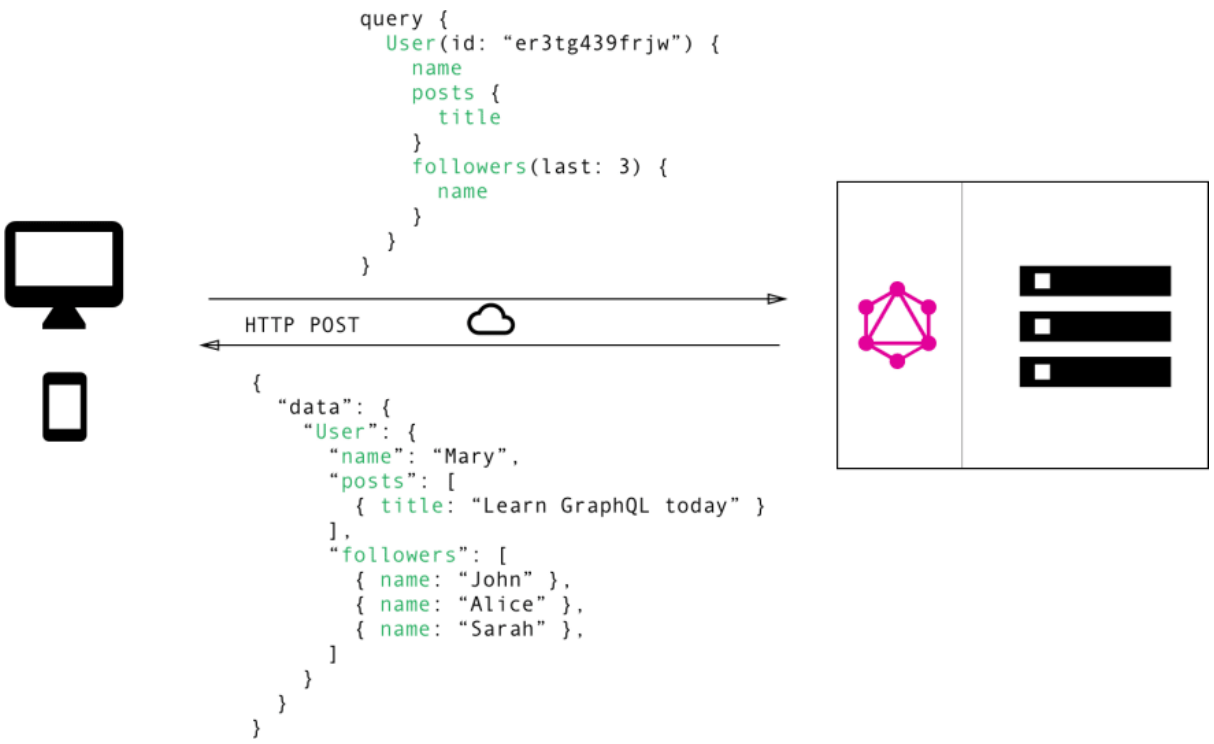
No underfetching

Isto se remete ao fato de quando um endpoint não fornece informações suficientes. Já com GraphQL não é necessário implementar vários endpoints para obter os dados necessários. Vamos observar a Figura 8:
Figura 8: Underfeching

Fonte: HowtoGraphQL, 2021

Quando você trabaha com REST, se você precisasse buscar um usuário pelo ID, você criaria um endpoint do tipo: /user/id. Perfeito! Agora se você precisasse dos posts deste usuário. Seguindo as normas REST você faria: -user/id/posts. Entendeu a ideia? Você não tem tudo o que precisa em um só endpoint. Caso você precise de mais detalhes, é necessário ficar criando novas rotas para suprir esta necessidade. Com o GraphQL você não passa por este problema. Um dos motivos, conforme foi dito no início deste artigo, é que a requisição é feita somente a um único endpoint. A partir dele, você requisita o que você precisa. Este exemplo pode ser visualizado na Figura 9:

Figura 9: No underfetching



No overfetching

Isto diz respeito ao fato de que o cliente faz download de mais dados do que o necessário na request para seu cliente. Em GraphQL? Tchauzinho sobrecarga.

Exemplo: hipoteticamente, você tem uma tela em que lista os dados de um produto. Para esta tela, você precisaria apenas de descrição do produto, valor e quantidade. No REST, você faria um endpoint mais ou menos assim: /products/id. Bacana. Aí em outra tela, você precisa de todos os dados deste produto. O que você faria? Requisitaria a mesma rota? Ok. Mas veja bem, na segunda tela você precisa de todos os dados do produto e faz sentido você transferir todos estes dados. Na primeira tela não. Não faz sentido sua API mandar para o cliente todos os dados do produto se você somente está precisando de tres campos. Se observarmos a Figura 9, com o GraphQL, você pode ir pedindo apenas o que você precisa. No REST, a solução para isto, ou você continua enviando os dados e filtra apenas o que precisa no front ou você cria 2 rotas, uma para todos os dados outra apenas para os 3 dados. Começa a fica esquisito, correto?

Prototipagem

Uma vez que GraphQL é flexível e trabalha somente com um endpoint, realizar uma prototipagem pelo menos ficou bem escalável. Repare que você não precisaria de ajustes no backend quando a estrutura de dados precisa ser mudada. No REST, é comum você construir endpoints tomando como base as views da aplicação.

Type System

O GraphQL, como vimos, possui um sistema de tipos forte, que são expostos e que servem parecido como um contrato entre cliente e servidor para especificar como os dados podem ser acessados. O legal do esquema é que ele também pode servir como documentação de sua API.

Bibliotecas

Diversas linguagens suportam GraphQL. Na própria documentação é possível ver estas linguagens, desde bibliotecas para frontend quanto backend. Seguem alguns exemplos:

- Go: graphql-go, 99designs/gqlgen, graphql-relay-go
- Java: graphql-java
- JavaScript: GraphQL.js, express-graphql, apollo-server
- PHP: graphql-php, Siler
- Python: graphene

CONSIDERAÇÕES FINAIS

Ufa galera!! Chegamos ao fim deste artigo. Acredito que foi bastante conteúdo e pode parecer impossível de aprender, mas não é. Obviamente, utilizar uma nova tecnologia a princípio, leva um tempo para você ir se acostumando. Mas garanto a vocês que GraphQL trouxe uma nova forma de construir APIs de forma espetacular.

Eu espero muito ter contribuído para o enriquecimento do conhecimento de vocês. Como falei durante este artigo, não vou parar por aqui. No próximo, mostrarei de maneira prática
de como podemos construir nossa API GraphQL utilizando a linguagem Golang. Quando a mesma estiver concluída, deixarei como referência aqui para vocês. Muito obrigado por terem lido até aqui e ajudaria muito o feedback de vocês. Até a próxima!!!!

CACHING

Client Side

O caching no lado do cliente refere-se à prática de armazenar temporariamente recursos (como imagens, arquivos de estilo, scripts ou dados) no dispositivo do usuário para reduzir a latência e melhorar o desempenho durante visitas subsequentes ao mesmo site ou aplicativo. Aqui estão os principais aspectos relacionados ao caching no lado do cliente:

Objetivo do Caching no Lado do Cliente:

- O objetivo principal do caching no lado do cliente é melhorar a experiência do usuário, reduzindo o tempo de carregamento de páginas e recursos. Ao armazenar localmente recursos que não mudam frequentemente, o navegador pode evitar fazer solicitações ao servidor a cada visita.

Tipos de Recursos Cachados:

- Imagens: Logotipos, fotos e gráficos.
- Arquivos de Estilo: Folhas de estilo CSS.
- Scripts: Arquivos JavaScript.
- Fontes: Arquivos de fonte.
- Dados: Respostas de solicitações AJAX ou dados JSON.

HTTP Cache Headers:

- A comunicação entre o servidor e o cliente sobre caching é regulada por cabeçalhos HTTP. Alguns cabeçalhos importantes incluem:
    - Cache-Control: Define diretivas para controle de caching.
    - Expires: Especifica uma data de validade para o recurso.
    - ETag: Uma tag de entidade única para verificar se o recurso mudou.
    - Last-Modified: Indica a última vez que o recurso foi modificado.

Cache-Control Directives:

- public: O recurso pode ser armazenado em cache pelo navegador e por servidores intermediários.
- private: O recurso só pode ser armazenado em cache pelo navegador do usuário.
- max-age: Define o tempo máximo, em segundos, que o recurso pode ser considerado válido.
- no-cache: O navegador deve revalidar o recurso com o servidor antes de utilizá-lo.
- no-store: O recurso não deve ser armazenado em cache.

LocalStorage e SessionStorage:

- O LocalStorage e o SessionStorage são mecanismos de armazenamento no lado do cliente que permitem armazenar dados no navegador. Eles podem ser usados para armazenar dados persistentes ou temporários entre as sessões do usuário.

Service Workers:

- Service Workers são scripts em execução em segundo plano que podem interceptar e manipular solicitações de rede. Eles podem ser usados para implementar estratégias avançadas de caching, como caching offline.

Bibliotecas de Caching no Lado do Cliente:

- Algumas bibliotecas e frameworks oferecem soluções específicas para caching no lado do cliente. Por

exemplo, o Workbox é uma biblioteca para facilitar a criação de Service Workers e caching offline em aplicações web.

Estratégias de Caching:

- Cache-First: O navegador verifica o cache antes de fazer uma solicitação à rede.
- Network-First: O navegador tenta obter o recurso da rede e, se falhar, recorre ao cache.
- Network-Only: O navegador ignora completamente o cache e busca o recurso na rede.
- Cache-Only: O navegador ignora a rede e usa apenas o cache.

Invalidação de Cache:

- É importante ter estratégias de invalidação de cache para garantir que os usuários obtenham versões atualizadas dos recursos quando necessário. Isso pode ser feito através da alteração dos cabeçalhos de controle de cache ou do uso de técnicas como "cache busting".

Cache Bust (Cache Busting):

- O cache busting envolve a alteração deliberada dos URLs dos recursos para evitar o uso de versões antigas armazenadas em cache. Isso pode ser feito adicionando um número de versão ao URL ou usando outras técnicas para garantir que o navegador busque sempre a versão mais recente.

Considerações de Segurança:

- Caching no lado do cliente deve ser feito com cuidado para evitar vazamento de dados sensíveis ou problemas de segurança. Certifique-se de que dados confidenciais não sejam armazenados em cache e implemente boas práticas de segurança.

Monitoramento e Relatórios:

- O desempenho do caching no lado do cliente deve ser monitorado e avaliado regularmente. Ferramentas de desenvolvedor do navegador e serviços de análise podem ajudar a identificar problemas e otimizações potenciais.

Implementar uma estratégia eficaz de caching no lado do cliente pode resultar em melhorias significativas no desempenho e na experiência do usuário. No entanto, é importante equilibrar a eficiência do caching com a necessidade de fornecer sempre versões atualizadas e precisas dos recursos.

CDN (Rede de fornecimento de conteúdo)

CDN (Content Delivery Network) é uma Rede de Distribuição de Conteúdo é um grupo de servidores que permitem que os conteúdos da internet estejam facilmente disponíveis, com rapidez e segurança. A rede é a responsável por melhorar a experiência do usuário enquanto usa os recursos dela de forma eficiente.

Algumas marcas conhecidas de CDN são Cloudflare, Akamai, Incapsula e MaxCDN. Na Hostinger, todos os nossos pacotes de hospedagem e planos WordPress têm suporte integral ao Cloudflare. E você pode usá-lo tranquilamente no seu site sem qualquer complicação.
Até dezembro de 2018, o número de usuários de internet atingiu um total de 4.1 bilhões. A maioria deles, senão todos, quer conteúdo de qualidade e da maneira mais rápida possível. Então, saber como trabalhar com os conteúdos do seu site é fundamental para se dar bem.

Vamos Entender o Que é Conteúdo?

Antes de nos aprofundarmos na forma como os conteúdos são entregues, precisamos detalhar o que é conteúdo.
Em suma, conteúdo é qualquer tipo de elemento textual, visual ou auditivo de um site. É um bloco de texto, uma imagem, arquivos de áudios, vídeos e assim por diante.

Existem dois tipos de conteúdos: o ESTÁTICO e o DINÂMICO. O conteúdo estático é aquele em que sua versão original (a de entrada) é o que as pessoas realmente veem na página publicada (o resultado). Simplificando, ele permanece o mesmo e não é modificado.
O servidor dá o mesmo tipo de dado a cada usuário; por conta disso, a entrega do conteúdo é bem mais rápida. O processo é simples: um usuário faz um pedido ao servidor de internet para acessar um arquivo e, na mesma hora, o servidor entrega o arquivo solicitado.
Já o conteúdo dinâmico é qualquer conteúdo que se modifica baseado na sua versão original (a de entrada). Ele é personalizado para várias páginas, dependendo das solicitações do usuário.

Um exemplo de conteúdo dinâmico é uma página de um produto: ela geralmente contém o nome do produto, uma descrição, um preço e inclui imagens. Outro exemplo é uma página que mostra informações relevantes ou registros das interações dos usuários.

Como Funciona uma CDN?

Basicamente, uma CDN reproduz o conteúdo que está armazenado no servidor central. A versão reproduzida do conteúdo vai, então, ser salva em locais de várias regiões do planeta, chamados de Pontos de Presença (PoP, do inglês). Esses Pontos são os locais em que mais de duas redes fazem a conexão consigo mesmas.

Sem uma CDN, quando um usuário tenta acessar um site, o computador envia um pedido de acesso ao servidor central para o conteúdo. O servidor central, então, responde ao pedido e mostra o conteúdo ao usuário.
Esse processo leva algum tempo para ser concluído. A distância entre o usuário e o servidor é que determina a velocidade do processo.

Por exemplo, vai levar um tempo para um usuário localizado no Rio de Janeiro conseguir uma resposta de um servidor localizado em Nova York (EUA).

Mas com a CDN, o negócio é diferente. Em vez de levar o pedido até o servidor central, os visitantes de um site recebem uma cópia dos dados (conteúdos já armazenados) do servidor de internet mais próximo.
A entrega do conteúdo, neste caso, é bem mais rápida e usa um sistema de mapeamento que avalia a localização do usuário e do servidor.

Se em alguma ocasião o servidor mais próximo não conseguir entregar o conteúdo ao usuário, ele vai procurar por outros servidores em um ambiente de CDN.
Além disso, se os dados não existem ou não tiverem sido armazenados, o servidor vai contactar o servidor central para fornecer tal conteúdo. E, então, vai armazená-lo para atender aos pedidos futuros.

Em resumo, um CDN tem data centers localizados em várias regiões do mundo. Sendo assim os PoPs consistem em um aglomerado de centenas de servidores.
E esses servidores trabalham sem cansar para acelerar o processo de entregar o conteúdo ao usuário, conforme sua solicitação.

4 Benefícios de Usar CDN

Por que você precisa de uma CDN? Existem 4 razões bem fundamentadas da importância de usar uma CDN. São elas: velocidade, uso amplo da internet, custo e segurança.

Melhorias na Velocidade

Cada segundo conta. Se um usuário de internet tenta acessar seu site e levar mais de 3 segundos para ver o conteúdo dela, é provável que ele desista do seu site.
Normalmente, o servidor de hospedagem do seu site é que faz o trabalho de gerenciar os dados e o tráfego de uma página. .
Mas, quando a quantidade de acessos sobe muito e você tem apenas um servidor para administrar tudo, uma resposta bem lenta é inevitável.
É aqui onde a CDN entra. Ele ajuda a responder as solicitações de acesso a uma página com muito mais velocidade. Como isso? Driblando a latência (o atraso entre o momento do pedido e a resposta) de conexões.
Alguns fatores podem causar esse atraso, mas é mais influenciado pela distância entre os visitantes e o servidor de hospedagem de um site.

Melhor Disponibilidade do Conteúdo

A quantidade de usuários de internet está crescendo rapidamente. Se você já pensava que fornecer conteúdos de qualidade poderia fazer explodir suas conversões, você está absolutamente certo.
Mas, para isso, você precisa se certificar que esses mesmos conteúdos estejam disponíveis em todas as ocasiões. Nesse caso, usar a CDN é uma ótima opção.

Imagine se seu site não funcionar corretamente quando o tráfego aumentar. Isso não apenas vai diminuir a credibilidade dele. Mas, também, será uma grande perda de oportunidade para mais conversões. A CDN foi feito para lidar com essas situações.

Custo-Benefício

Quem é que não quer ter mais resultados gastando menos? Com a CDN, você pode ter um projeto com um melhor custo-benefício. Quando menos dados são necessários de um servidor de origem, os custos de hospedagem podem ser ajustados de acordo com a demanda.
Isso significa que você não precisa pagar altas quantias de dinheiro para ter um serviço eficiente e que você mesmo pode otimizar. Outra coisa boa é que você não precisa criar uma infraestrutura. A empresa de CDN é que cuida disso para você.

Segurança é Tudo

A segurança do seu site e dos dados dele devem ser a sua prioridade número 1. Tanto na sua perspectiva quanto na dos seus usuários.
Mas, na maioria das vezes, você não precisa realmente saber como se proteger de todos os ataques maliciosos. O fato é que um ataque hacker acontece a cada 39 segundos e 95% das brechas em segurança digital são causadas por falha humana.
Além disso, acredito que você saiba o quão destrutivo um ataque DDoS pode ser. É uma tentativa maliciosa de perturbar um servidor ou uma rede com uma avalanche de pedidos e tráfego desorientado.
Como resultado, seu site cai e não pode ser acessado por ninguém. Qualquer negócio que você tiver na internet corre risco se algo desse tipo acontecer. Seus consumidores podem, também não querer mais acessar seu site e procurarem opções mais seguras.

Ainda, se você já implementou uma CDN, é ele quem vai gerenciar todo o tráfego recebido pelo site, garantindo que este esteja sempre funcionando. A rede estará lá para ajudar você a bloquear um ataque malicioso antes que ele atinja seu data center ou servidor.
Agora, que tipos de negócios realmente precisam de uma CDN (Rede de Distribuição de Conteúdo)? Abaixo listamos alguns exemplos de negócios em que ter um CDN faz toda a diferença.

- **E-Commerce**
  Falar sobre E-Commerce é falar sobre receber tráfego de todos os cantos do mundo. Os produtos e serviços existem em uma variedade gigantesca, mas os consumidores é que sempre têm um tempo limitado para se decidirem por uma compra. Se um site falha em fornecer a informação necessária naquele momento, oportunidades incalculáveis são perdidas num piscar de olhos. Um E-Commerce precisa de CDN para lidar com a quantidade de pedidos muitos locais diferentes. È aqui onde a CDN mostra suas vantagens. Armazenando o conteúdo no servidor mais próximo e entregando uma resposta mais rápida, o CDN ainda previne picos indesejados de tráfego para que o servidor não fique sobrecarregado.
- **Publicidade**
  A publicidade digital atualmente usa anúncios baseados em conteúdos multimídia para engajar e vender seus produtos. Eles são mais atraentes, informativo e interativos. Mas, aqui aparece o problema: eles demandam muitos recursos para funcionar bem. Em um cenário caótico teríamos um site abarrotado com publicidade multimídia carregando bem lentamente. Quanto mais lento é um site, mais os consumidores tendem a abandoná-lo. Um negócio de publicidade precisa de um CDN para resolver esse problema. Como a CDN armazena conteúdos em cache no servidor mais próximo, todos os conteúdos são carregados bem mais rapidamente. O tempo mínimo de carregamento pode ser mantido e o desempenho do site será bem melhor.
- **Jogos Online**
  Se a publicidade precisa de muitos recursos para conteúdo, os jogos eletrônicos precisam de muito mais recursos. Esse é o maior desafio para a indústria de games: continuar entregando o melhor conteúdo, mas ao mesmo tempo, driblar os problemas de desempenho nos games. A tecnologia CDN permite que os jogos online tenham as chamadas “zonas de poder”, um lugar em que os desenvolvedores podem hospedar um game inteiro em um servidor CDN. Nesse cenário, a necessidade de fazer um

pedido diretamente para o servidor de origem praticamente desaparece.

- **Entretenimento**
  O conteúdo é o coração da indústria de mídia e do entretenimento. Dos downloads aos streamings, os conteúdos de entretenimento atraem milhões de pessoas mundo afora. Donos de sites que trabalham com estes tipos de conteúdos têm uma estratégia sólida para manter seus sites sempre funcionando perfeitamente. Novamente, dados salvos em cache pela CDN é o salvador de uma situação catastrófica. As cópias do conteúdos salvas em múltiplos servidores estarão prontas para atender aos pedidos dos usuários baseados na localização deles. E isso com certeza acelera o desempenho de um serviço baseado em conteúdos.

Qual é a Diferença Entre CDN e VPN?

A velocidade de entrega de informação das CDNs se dá ao enviar e/ou armazenar conteúdo dos websites através de uma rede de servidores, permitindo que os visitantes acessem páginas da web a partir da fonte mais próxima disponível.
Já as Virtual Private Networks (VPNs) protegem a identidade do usuário e usam uma série de servidores em diferentes localidades para contornar restrições geográficas ou de outras naturezas em certos tipos de conteúdo. Ambos proporcionam segurança extra e melhoram o acesso a conteúdos, mas para diferentes propósitos.

Conclusão

Uma CDN (Content Delivery Network ou Rede de Distribuição de Conteúdo) é importante tanto para os usuários quanto para quem tem um site. Você pode usar uma CDN se seu negócio precisar de carregamentos mais rápidos, além de melhores disponibilidade de conteúdo e custo-benefício. Assim, tráfego muito grande não

será um problema. Apenas foque no seu conteúdo e não esqueça da qualidade!

REDIS

Os dados são a base de todas as iniciativas de transformação digital no setor financeiro, do qual os consumidores esperam alto desempenho em tempo real. Afinal, as oportunidades desaparecem tão rapidamente quanto aparecem.

E uma ferramenta completa hoje, precisa não apenas contar com um banco de dados de alto desempenho, como também fornecer tempos de resposta abaixo de milissegundos, armazenar dados de dezenas de fontes, entregar alta disponibilidade e segurança em várias camadas.
Nesse sentido, o Redis é a ferramenta perfeita para resolver esses problemas e ajudar a aumentar a competitividade no setor financeiro.
Quer saber como essa tecnologia funciona e qual seu impacto no mercado de ações? Então continue neste artigo para saber mais!

O que é REDIS?

O Redis (Remote Dictionary Server) é um tipo de banco de dados que armazena as informações em código aberto, altamente replicado, de alto desempenho e não relacional. Como armazena os dados na memória, a tecnologia é mais usada como cache.
Algumas grandes organizações que usam Redis são Twitter, GitHub, Instagram, Pinterest e Snapchat.

O Redis oferece suporte a strings, baseadas em valores-chave e que são mapeados para armazenar e recuperar informações. No entanto, também consegue suportar outras estruturas de dados complexas como, por exemplo, listas e conjuntos.

Esse suporte amplo a diferentes tipos de dados aproxima o Redis das estruturas de dados nativas que os programadores mais comumente usam.
Essa facilidade de uso torna o Redis excelente para o desenvolvimento de aplicativos, isso porque as principais estruturas de dados são compartilhadas de forma descomplicada entre processos e serviços. De acordo com Salvatore Sanfilippo, criador do Redis, “O Redis é um banco de dados em memória que pode ser usado como cache, banco de dados e broker de mensagens. Ele é adequado para aplicativos que requerem alta velocidade de leitura/gravação e escalabilidade horizontal.”

Quais os benefícios de usar a tecnologia REDIS?

Uma das principais vantagens de utilizar a tecnologia Redis é a sua velocidade. Como o seu armazenamento é feito na memória, ele pode executar 110.000 SETs e 81.000 GETs por segundo.
Mas, além desse benefício, existem outras vantagens que merecem ser destacadas, como:

- **Estruturas de dados flexíveis:** o Redis oferece suporte a diversos tipos de dados avançados, sendo os principais: strings, listas, conjuntos, conjuntos ordenados, hashes, bitmaps, hyperloglogs, lists, dentre outros;
- **Alta performance:** a tecnologia possui baixa latência e alta taxa de transferência ao acessar os dados;
- **Fácil configuração:** o Redis possui uma estrutura de comando simples o que torna a configuração mais fácil;
- **Maior performance na leitura:** a tecnologia consegue distribuir as solicitações entre vários servidores ao mesmo tempo, o que melhora a leitura das informações ;
- **Suporte a backups:** os dados do Redis, por padrão, são salvos em um único arquivo compacto point-in-time (chamado de RDB – Redis Database) em intervalos especificados. Isso permite restaurar facilmente diferentes versões do conjunto de dados em caso de desastres.
- **Versatilidade:** a tecnologia pode ser usada em conjunto com outros bancos de dados, oferecendo suporte para

reduzir a carga e melhorar o desempenho. Também pode ser usado como banco de dados primário;
- **Maior alcance de transações:** o Redis suporta diferentes tipos de transações (MULTI, EXEC, DISCARD e WATCH são as principais) para operações que envolvem vários comandos;
- **Oferece conceitos de replicação para alta disponibilidade:** em caso de falha do servidor primário, um banco de dados secundário garante o funcionamento do programa;
- **Fragmentação para dimensionamento horizontal:** os dados são divididos em buckets menores e independentes, para permitir a adição de novos conforme necessário.

Em resumo, o Redis possui as funcionalidades ideias para aplicações que necessitem de dinamismo em seu processo, com acesso frequente a dados essenciais.

Tecnologia REDIS no mercado financeiro

O mercado financeiro está constantemente investindo em tecnologia para melhorar a experiência dos clientes e otimizar a tomada de decisão.
O Redis, portanto, pode ajudar nesse sentido, já que fornece o necessário para a construção de plataformas mais responsivas e seguras. Sendo
Além de melhorar a experiência dos clientes e agilizar a tomada de decisão, o Redis é capaz de potencializar os serviços financeiros de outras formas, tais como:

1- Detecção e mitigação de fraudes

O Redis permite que as empresas examinem padrões em históricos de transações para buscar por comportamentos suspeitos ou fraudulentos.

2- Engajamento e personalização para o cliente

O Redis oferece vários modelos de dados necessários para manter os perfis dos clientes atualizados em tempo real, o que permite:

- Análise de risco de crédito;
- Publicidade direcionada;
- Promoções de cartão de crédito e muito mais.

3- Cache Inteligente

A ferramenta permite uma série de padrões de armazenamento em cache inteligentes. Isso permite que você escolha:

- Por quanto tempo deseja manter os dados;
- Ordem de remoção de informações;
- Definir os padrões de armazenamento.

4- Script Lua

O Redis possui uma funcionalidade que possibilita que scripts personalizados sejam escritos e executados na linguagem Lua. Isso possibilita que os usuários adicionem recursos à ferramenta na forma de scripts de execução rápida.

5- Replicação de banco de dados ativo

Como permite a replicação de banco de dados ativo, é possível lidar com atualizações simultâneas de várias localizações geográficas. Potencializando, assim, a detecção de fraude, limitação de taxa e personalização.

6- Segurança de nível empresarial

O Redis garante que os dados sejam isolados, o que oferece um tipo de segurança multicamadas para controle de acesso, autenticação e autorização.

7- Integrações de provedor de nuvem e plataforma

A tecnologia também pode ser integrada a diversas plataformas e provedores de nuvem. O que garante automação e suporte para realizar tarefas operacionais.

No mercado financeiro, segundos são suficientes para impactar os resultados positivamente ou negativamente. O Redis, portanto, pode otimizar o acesso à informação e cotações de ativos, em tempo real.

Além disso, o Redis reduz a complexidade de sua tecnologia, diminui os custos, melhora a capacidade de resposta ao cliente, e aumenta a segurança das transações financeiras.
Tudo isso em uma única plataforma. Invista em tecnologia e transforme seu negócio.Conheça as soluções da Cedro.

CONHECIMENTO SOBRE WEB SECURITY

HASHING ALGORITHMS

- ### MD5 E COMO USÁ-LO

O **algoritmo de sintetização de mensagem MD5** é uma função hash amplamente utilizada que produz um valor de hash de 128 bits expresso em 32 caracteres. Embora o MD5 tenha sido projetado inicialmente para ser usado como uma função hash criptográfica, foi constatado que ele sofre de extensas vulnerabilidades. Ele ainda pode ser usado como uma soma de verificação para checar a integridade de dados, mas apenas contra corrupção não intencional. Ele permanece adequado para outros fins não criptográficos, por exemplo, para determinar a partição para uma chave específica em um banco de dados particionado.

O MD5 foi projetado por Ronald Rivest em 1991 para substituir uma função hash anterior MD4, e foi especificado em 1992 como RFC 1321.

Um requisito básico de qualquer função de hash criptográfico é que seja computacionalmente inviável encontrar duas mensagens distintas com hash para o mesmo valor. O MD5 falha com esse

requisito catastroficamente; essas colisões podem ser encontradas em segundos em um computador doméstico comum.

As fraquezas do MD5 foram exploradas em campo, principalmente pelo malware Flame em 2012. O CMU Software Engineering Institute considera o MD5 essencialmente "criptograficamente quebrado e inadequado para uso posterior". Em 2012, o malware Flame explorou os pontos fracos do MD5 para falsificar uma assinatura digital da Microsoft.

Em 2008, Ronald Rivest e outros, publicaram uma nova versão do algoritmo o MD6 com hash de tamanhos 224, 256, 384 ou 512 bits. O algoritmo MD6 iria participar do concurso para ser o novo algoritmo SHA-3, porém logo depois removeu-o do concurso por considerá-lo muito lento, anunciando que os computadores de hoje são muito lentos para usar o MD6.

A partir de 2019, o MD5 continua sendo amplamente utilizado, apesar de suas fraquezas e depreciação bem documentadas por especialistas em segurança.

Vulnerabilidade

Como o MD5 faz apenas uma passagem sobre os dados, se dois prefixos com o mesmo hash forem construídos, um sufixo comum pode ser adicionado a ambos para tornar uma colisão mais provável. Deste modo é possível que duas strings diferentes produzam o mesmo hash. O que não garante que a partir de uma senha codificada em hash específica consiga-se a senha original, mas permite uma possibilidade de descobrir algumas senhas a partir da comparação de um conjunto grande de hash de senhas através do método de comparação de dicionários.

Pseudocódigo

Segue-se um pseudocódigo para o algoritmo MD5

```
//Definir r como o seguinte
var int[64] r, k
```

```
r[ 0..15] := {7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22}
r[16..31] := {5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20}
r[32..47] := {4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23}
r[48..63] := {6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21}

//Utilizar a parte inteira dos senos de inteiros como constantes:
for i from 0 to 63
    k[i] := floor(abs(sin(i + 1)) × 2^32)

//Iniciar as variáveis:
var int h0 := 0x67452301
var int h1 := 0xEFCDAB89
var int h2 := 0x98BADCFE
var int h3 := 0x10325476

//Pre-processamento:
append "1" bit to message
append "0" bits until message length in bits ≡ 448 (mod 512)
append bit length of message as 64-bit little-endian integer to message

//Processar a mensagem em pedaços sucessivos de 512-bits:
for each 512-bit chunk of message
     break chunk into sixteen 32-bit little-endian words w(i), 0 ≤ i ≤ 15

   //Inicializar o valor do hash para este pedaço:
    var int a := h0
    var int b := h1
    var int c := h2
    var int d := h3
```

```
   //Loop principal:
    for i from 0 to 63
        if 0 ≤ i ≤ 15 then
            f := (b and c) or ((not b) and d)
            g := i
        else if 16 ≤ i ≤ 31
            f := (d and b) or ((not d) and c)
            g := (5×i + 1) mod 16
        else if 32 ≤ i ≤ 47
            f := b xor c xor d
            g := (3×i + 5) mod 16
        else if 48 ≤ i ≤ 63
            f := c xor (b or (not d))
            g := (7×i) mod 16

       temp := d
        d := c
        c := b
        b := ((a + f + k[i] + w(g)) leftrotate r[i]) + b
        a := temp

   //Adicionar este pedaço do hash ao resultado:
    h0 := h0 + a
    h1 := h1 + b
    h2 := h2 + c
    h3 := h3 + d

var int digest := h0 append h1 append h2 append h3
//(expressed as little-endian)
```

Nota: Ao invés da formulação do RFC 1321 acima exibida, considera-se mais eficiente a seguinte implementação:

```
(0  ≤ i ≤ 15): f := d xor (b and (c xor d))
(16 ≤ i ≤ 31): f := c xor (d and (b xor c))
```

Hashes MD5

Os hashes MD5 de 128-bit (16-byte) são normalmente representados por uma sequência de 32 caracteres hexadecimais. O seguinte mostra uma string ASCII com 43-bytes e o hash correspondente:

```
MD5("The quick brown fox jumps over the lazy dog")
  = 9e107d9d372bb6826bd81d3542a419d6
```

Mesmo uma pequena alteração na mensagem vai criar um hash completamente diferente, ex. ao mudar d para c:

```
MD5("The quick brown fox jumps over the lazy cog")
  = 1055d3e698d289f2af8663725127bd4b
```

O Hash de uma string vazia é:

```
MD5("")
  = d41d8cd98f00b204e9800998ecf8427e
```

Salgar (Salting)

Para aumentar a segurança em alguns sistemas, usa-se a tática de adicionar um texto fixo no texto original a ser codificado. Deste modo se o sal for "wiki" e a senha for "1234", a pseudo-senha poderá ser "wiki1234" e assim mesmo que alguém tenha o MD5 de 1234 por ser uma senha comum ele não terá de wiki1234. Porém caso o "sal" seja simples como no exemplo e houver o MD5 de "wiki1234" é possível descobrir o sal e deste modo decodificar as senhas mais comuns. Por este motivo geralmente o "sal" é algo complexo.

- # S.H.A FAMILY (SECURE HASH ALGORITHM)

Secure Hash Algorithms (em português, Algoritmos de Hash Seguro) são uma família de funções hash criptográficas publicadas pelo Instituto Nacional de Padrões e Tecnologia (NIST) como um Federal Information Processing Standard (FIPS) dos EUA, incluindo:[1]

SHA-0: um retrônimo aplicado à versão original da função de hash de 160 bits publicada em 1993 sob o nome "SHA". Foi retirado logo após a publicação devido a uma "falha significativa" não revelada e substituído pela versão ligeiramente revisada SHA-1.

SHA-1: Uma função hash de 160 bits que se assemelha ao algoritmo MD5 anterior. Foi desenvolvido pela Agência de Segurança Nacional (NSA) para fazer parte do algoritmo de assinatura digital. As fraquezas criptográficas foram descobertas no SHA-1 e o padrão não era mais aprovado para a maioria dos usos criptográficos após 2010.

SHA-2: Uma família de duas funções de hash semelhantes, com tamanhos de bloco diferentes, conhecidas como SHA-256 e SHA-512. Eles diferem no tamanho da palavra; O SHA-256 usa palavras de 32 bytes, enquanto o SHA-512 usa palavras de 64 bytes. Também existem versões truncadas de cada padrão, conhecidas como SHA-224, SHA-384, SHA-512/224 e SHA-512/256. Estes também foram projetados pela NSA.

SHA-3: Uma função de hash chamada Keccak, escolhida em 2012 após uma competição pública entre designers não pertencentes à NSA. Ele suporta os mesmos comprimentos de hash do SHA-2 e sua estrutura interna difere significativamente do restante da família SHA.

Os padrões correspondentes são FIPS PUB 180 (SHA original), FIPS PUB 180-1 (SHA-1), FIPS PUB 180-2 (SHA-1, SHA-256, SHA-384 e SHA-512). O NIST atualizou a Publicação de Projeto FIPS 202, Padrão SHA-3 separada do Secure Hash Standard (SHS).

- SCRYPT

Scrypt é uma função hash criptográfica que foi originalmente projetada para uso como uma função de derivação de chaves a partir de senhas no Tarsnap. Ela foi formalizada em um artigo científico e apresentada por Colin Percival em maio de 2009 na conferência BSDCan'09[1] e teve sua primeira implementação lançada no dia 08 do mesmo mês. Um paper posterior sobre scrypt foi publicado no Internet Engineering Task Force (IETF) como um Request For Comments (RFC) de número 7914 em Agosto de 2016

O desenvolvimento da função scrypt ocorre de maneira open-source no seu repositório do git, e tem implementações nas linguagens C, Go, Python, Haskell, Node e Ruby.

Ela é uma função que por design exige um uso de memória mais alto na sua computação em comparação a outras funções de derivação de chaves, como PBKDF2 e bcrypt. Ela foi projetada dessa forma para dificultar ataques de força bruta, onde um atacante conhece um valor hash H e deseja saber qual chave o gerou, para isso ele itera sobre uma possível lista de chaves e aplica a função hash sobre cada uma delas até que a saída da função seja igual a H.

Algoritmo
Entradas:

P: Palavra chave que será digerida pela a função, uma sequência de bytes.

S: Sal, uma sequência de bytes que têm como defender contra ataques de dicionário e contra ataques pré-computados como rainbow table.
N: Custo CPU/memória.
p: Parâmetro de paralelização, um inteiro positivo que satisfaz p ≤ (232- 1) * 32/ SMixLen.
dkLen: Tamanho da saída desejada em bytes, um inteiro positivo que satisfaz dkLen ≤ (232- 1) * 32.
Saídas:

DK: Chave derivada, com dkLen bytes

```
Function scrypt(P,S,N,p,dkLen):
    (B0 ... Bp-1) ← PBKDF2(HMAC_SHA256, P, S, 1, p * SMixLen)
    for i = 0 to p-1 do
        Bi ← SMix(Bi,N)
    end for
    DK← PBKDF2(HMAC_SHA256, P, B0 || B1 ... Bp-1, 1, dkLen)
```

Integerify() é uma função bijetiva de {0, 1}k para {0,...,2k-1}.

```
Function SMix(B,N):
    X ← B
    for i = 0 to N - 1 do
       Vi ← X
       X ← BlockMix(X)
    end for
    for i = 0 to N - 1 do
        j ← Integerify(X) mod N
        X ← BlockMix(X ⊕ Vj)
    end for
    Output ← X
Function BlockMix(B):
    (B0, ... , B2r-1) ← B
    X ← B2r-1
    for i = 0 to 2r - 1 do
        X ← H(X ⊕ Bi)
        Yi ← X
    end for
```

```
Output ← (Y0, Y2, ... , Y2r-2, Y1, Y3, ... , Y2r-1)
```

Versões
Todas as versões oficiais do projeto scrypt estão disponíveis para download no seu repositório do Github e no Tarsnap.

Versão	Data de Lançamento
scrypt 1.2.1	11/02/2017
scrypt 1.2.0	30/07/2015
scrypt 1.1.6	16/01/2010
scrypt 1.1.5	06/11/2009
scrypt 1.1.4	15/06/2009
scrypt 1.1.3	25/05/2009
scrypt 1.1.2	20/05/2009
scrypt 1.1.1	16/05/2009
scrypt 1.1	16/05/2009
scrypt 1.0	08/05/2008

Uso como prova de trabalho

A sua natureza de ser propositalmente custosa de ser computada várias vezes e fácil de ser computada apenas uma vez torna scrypt um bom candidato para uso na construção de uma prova de trabalho. Prova de trabalho é um pedaço de data que é difícil de ser produzida, mas é fácil de ser verificada e foi inicialmente criada para evitar spam de emails e ataques de negação de serviço. Por exemplo, antes que um remetente possa enviar um e-mail, é preciso que sua máquina realize uma computação que é pouco demorada se o usuário deseja enviar o e-mail para apenas um destinatário e muito demorada caso o remetente deseje enviar e-mail para vários destinatários. Essa técnica atribui um custo ao envio de e-mail para vários destinatários, efetivamente diminuindo o modelo do spam, que depende de enviar uma grande quantidade de emails de uma maneira barata.

O sistema Hashcash implementa essa prova de trabalho só permitindo que o destinatário receba o e-mail após verificar uma estampa válida do cabeçalho do email. No entanto, a única forma do remetente encontrar uma estampa válida é aplicar uma função hash sobre valores aleatórios até que o valor hash tenha uma certa quantidade de zeros.

Com o advento do Bitcoin o sistema Hashcash começou a ser utilizado em criptomoedas de modo a determinar qual dos nós mineradores pode sugerir o próximo bloco a ser adicionado a blockchain e consequentemente ser recompensado. No Bitcoin o sistema Hashcash é usado de uma análoga a forma que é utilizado em um sistema de emails. Um nó que esteja minerando precisa calcular um valor tal que ao ser processado por uma função de hash SHA-256 junto com o bloco que ele deseja sugerir tenha uma certa quantidade de zeros que todos da rede sabem. Um nó que achar tal valor, pode então transmitir para o resto da rede Bitcoin o bloco que encontrou e qualquer outro nó da rede pode verificar se o novo bloco é válido, aplicando a função hash SHA-256 sobre o bloco recebido e vendo se o resultado tem o número de zeros necessário.

Por ser uma função hash criptográfica, o scrypt pode ser usado como uma alternativa ao SHA-256 no processo de mineração de criptomoedas, e é de primeira vista interessante pelo seu custo elevado não só de cpu mas também de memória.

Criptomoedas que utilizam scrypt

O fato do scrypt, quando utilizado como prova de trabalho, aparentemente ser resistente à mineração utilizando placas de vídeo e circuitos de integrados de aplicação específica (ASICs) causada por um grande custo de memória na hora de sua computação[4], atraiu muitos desenvolvedores de novas criptomoedas. A partir do fim dos anos 2011, foram criadas várias criptomoedas utilizando scrypt, que buscavam fugir do algoritmo de hash SHA-256 utilizado pelo Bitcoin, que desde o

final de 2010 já tinha seu processo de mineração dominado por placas de videos[5].

Tenebrix

Foi a primeira criptomoeda criada a implementar um esquema de prova de trabalho baseado em scrypt[6]. Foi proposta em 26 de setembro de 2011 no fórum bitcointalk pelos membros “Lolcust” e “ArtForz”. Outro fato notável sobre essa criptomoeda é que ela foi lançada com cerca de 7 milhões de moedas pré mineradas.

Fairbrix

Foi proposta por Charlie Lee utilizando o apelido de “coblee” no fórum bitcointalk em 2 de outubro de 2011. Fairbrix foi um fork do Tenebrix, logo utilizava scrypt, e tinha como maior diferença um número mais reduzido de blocos pré-minerados.

Litecoin

Lançada em 9 de outubro de 2011 no fórum bitcointalk também por Charlie Lee. Litecoin teve a proposta de tentar melhorar o Bitcoin, reduzindo o tempo de chegada de blocos para dois minutos e meio, quadruplicando o número de moedas que podem ser criadas e utilizando um algoritmo diferente do SHA-256, utilizado pelo Bitcoin. Litecoin teve um grande sucesso no mercado de criptomoedas e chegou a ficar em segundo lugar no ranking de capitalização de mercado, ao lado do Bitcoin nos meados de 2014[7]. Em Agosto de 2017, Litecoin continua sendo desenvolvida como código aberto e ocupa quinta posição no ranking de capitalização de mercado com aproximadamente 2 bilhões e 500 milhões de dólares[8]. Litecoin utiliza uma versão simplificada da função scrypt sem a feature de alongamento de chave, a mineração utiliza os parâmetros de 1 iteração e 128Kb de uso de memória[9]. O motivo da escolha

desses parâmetros é que os desenvolvedores do Litecoin consideraram que o custo da verificação seria muito elevado, apesar de que parâmetros maiores iriam dificultar a produção de circuitos de integrados de aplicação específica (ASIC) para a mineração.

Dogecoin

É uma das muitas criptomoedas que foram lançadas após o sucesso da Litecoin, utilizando scrypt com os mesmos parâmetros estabelecidos pela Litecoin[11]. Foi anunciada no fórum bitcointalk em 8 de dezembro de 2013 e se destaca por utilizar o meme da internet "Doge" como seu logo.

Gridcoin

Ao ser criada em Outubro de 2013[12], Gridcoin utilizava um sistema de prova de trabalho utilizando o scrypt para manter a rede segura [13], atualmente essa versão que utilizava prova de trabalho é chamada de Gridcoin-Classic. Gridcoin visa recompensar mineradores que estão trabalhando na manutenção da rede e são voluntários do projeto BOINC, um projeto da Universidade de Berkeley que fornece uma plataforma para que voluntários possam fornecer o tempo ocioso de processamento para colaborar com pesquisas no ramos de medicina, biologia, matemática, climatologia e astrofísica. O maior benefício que Gridcoin queria oferecer era direcionar o poder computacional dos mineradores para projetos BOINC enquanto que operações de mineração ficariam em segundo plano.

Entre 12 de Outubro de 2014 e 26 de Abril do ano seguinte o sistema de prova de trabalho com scrypt utilizado pelo Gridcoin-Classic foi substituído totalmente, por causa do custo ainda elevado de energia, para um sistema de prova de pesquisa[14] chamado de Gridcoin-Research. O processo de migração que durou 6 meses, consistiu na queima de moedas que

foram transferidas para um endereço sem dono. Por fim, Gridcoin-Research se tornou Gridcoin e a blockchain do Gridcoin-Classic foi aposentada.

Variações do scrypt

A mineração da Litecoin que utilizava a versão original do scrypt, durou pouco tempo sendo feito através de gpu e cpu. Circuitos de integrados de aplicação específica (ASICs) para os parâmetros utilizados pelo Litecoin, e pelas outras inúmeras criptomoedas que a copiaram, começaram a ser produzidos a partir do fim de 2013 [15][16]. Esse evento pode ser explicado devido aos baixo valor do parâmetro N (que determina o uso de memória) utilizado pelas altcoins baseadas em Litecoin. Qualquer mudança no algoritmo de forma a dificultar uso de ASICs após sua produção em massa é difícil, pois para que o algoritmo seja mudado é preciso que a maioria do poder computacional da rede vote para tal, e a maioria do poder computacional estaria na mão dos mineradores que utilizam ASICs.

Graças a sua popularidade em uso como prova de trabalho, foram criadas outras variações do scrypt que procuraram tornar criptomoedas que o utilizavam resistentes à mineração utilizando circuitos de integrados de aplicação específica (ASICs).

Scrypt-N

Scrypt-N ou Scrypt Adaptive-N é um algoritmo, usado primeiramente pela Vertcoin, aonde diferentemente do scrypt utilizado pelo Litecoin que têm parâmetro N fixo utiliza o parâmetro de requisito de memória variável e incremental com o tempo[17]. A motivação inicial para uso de um N incremental era que o aumento da memória necessária para a computação da função iria dificultar a produção de ASICs, o que acabou não sendo verdade novamente com o possível desenvolvimento de

ASICs para Scrypt-N e a eventual troca de função hash na Vertcoin.

- BCRYPT

bcrypt é um método de criptografia do tipo hash para senhas baseado no Blowfish. Foi criado por Niels Provos e David Mazières e apresentado na conferência da USENIX em 1999.
Este método apresenta uma segurança maior em relação à maioria dos outros métodos criptográficos que é a implementação da variável "custo" que é proporcional à quantidade de processamento necessária para criptografar a senha. O método é conhecido como hash adaptativo às melhorias futuras de hardware por ter esta característica, pois pode permanecer resistente à ataques do tipo "força-bruta" com o tempo usando custos maiores de processamento.

Ao contrário do método tradicional "crypt" (concebido em 1976), o algoritmo bcrypt não possui as restrições da época que eram pouco processamento e poucos espaço (bytes) para guardar o salt e o hash gerado da senha criptografada.
O algoritmo bcrypt foi implementado em diversas linguagens como Python, Perl, Ruby, Java, C# e outras, além de possuir implementação também para a função "crypt" do UNIX.

API SECURITY - BOAS PRÁTICAS

- ## HTTPS

HTTPS (Hyper Text Transfer Protocol Secure - protocolo de transferência de hipertexto seguro) é uma implementação do protocolo HTTP sobre uma camada adicional de segurança que utiliza o protocolo TLS/SSL. Essa camada adicional permite que os dados sejam transmitidos por meio de uma conexão criptografada e que se verifique a autenticidade do servidor e do cliente por meio de certificados digitais. A porta TCP usada por norma para o protocolo HTTPS é a 443.

O protocolo HTTPS é utilizado, em regra, quando se deseja evitar que a informação transmitida entre o cliente e o servidor seja visualizada por terceiros, como por exemplo no caso de compras online. A existência na barra de endereços de um cadeado (que pode ficar do lado esquerdo ou direito, dependendo do navegador utilizado) demonstra a certificação de página segura (TLS/SSL). A existência desse certificado indica o uso do protocolo HTTPS e que a comunicação entre o browser e o servidor se dará de forma segura. Para verificar a identidade do servidor é necessário um duplo clique no cadeado para exibição do certificado.

Nas URLs dos sites o início ficaria https://.
Um exemplo de conexão via HTTPS é o próprio site da Wikimedia Foundation, em que é possível acessar e editar o conteúdo dos sites através de uma conexão segura. Através da URL https://secure.wikimedia.org/wikipedia/pt/wiki/Página_principal é possível editar a Wikipédia em língua portuguesa.

Conexões HTTPS são frequentemente usadas para transações de pagamentos na World Wide Web e para transações sensíveis em sistemas de informação corporativos. Porém, o HTTPS não deve

ser confundido com o protocolo "Secure HTTP" (S-HTTP), especificado na RFC 2660 e raramente utilizado.

Ilustração do protocolo de rede https e das letras www
HTTPS é um esquema URI, isto é, com exceção do esquema de tokens, é sintaticamente idêntico ao esquema HTTP utilizado para conexões normais HTTP, mas sinaliza ao navegador para utilizar uma camada adicional de criptografia utilizando o protocolo TLS (ou seu antecessor SSL) e para proteger o tráfego. TLS é especialmente adequado a HTTP porque pode fornecer proteção mesmo se apenas uma das partes comunicantes esteja autenticada. Este é o caso das transações HTTP na Internet, em que tipicamente apenas o servidor está autenticado, através da verificação de seu certificado realizada pelo cliente.

A ideia principal do HTTPS é criar um canal seguro sobre uma rede insegura. Isso garante uma proteção razoável de pessoas que realizam escutas ilegais (os chamados eavesdroppers) e de ataques de homem-no-meio (man-in-the-middle), dado que a cifragem foi adequadamente utilizada e que o certificado do servidor é verificável e confiável.

A confiança fornecida pelo HTTPS é baseada em autoridades de certificação que vêm pré-instaladas no navegador (isto é equivalente a dizer "Eu confio na autoridade de certificação VeriSign/Microsoft/etc. para me dizer em quem devo confiar"). Portanto, uma conexão HTTPS pode ser confiável se e somente se todos os itens a seguir são verdade:

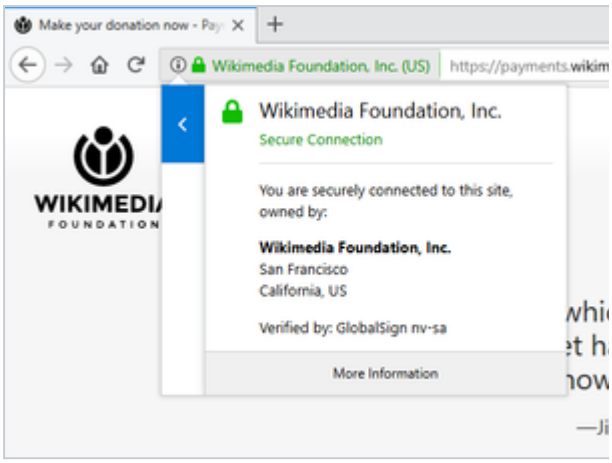
1. O usuário confia que o navegador implementa corretamente HTTPS com autoridades certificadoras pré-instaladas adequadamente;
2. O usuário confia que as autoridades verificadoras só irão confiar em páginas legítimas, que não possuem nomes enganosos;
3. A página acessada fornece um certificado válido, o que significa que ele foi assinado por uma autoridade de certificação confiável;
4. O certificado identifica corretamente a página (por exemplo, quando o navegador acessa "https://exemplo.com", o certificado recebido é realmente de "Exemplo Inc." e não de alguma outra entidade);
5. Ou o tráfego na internet é confiável, ou o usuário crê que a camada de encriptação do protocolo SSL/TLS é suficientemente segura contra escutas ilegais (eavesdropping).

Utilização nos websites

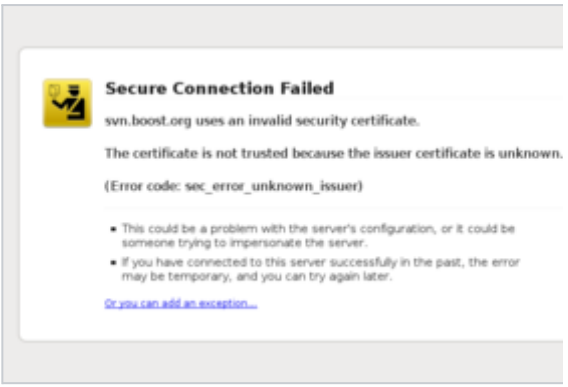
Em 2017, o HTTPS era utilizado por somente 9,69% do total de domínios brasileiros registrados e 14,51% dos domínios portugueses.

Integração com o navegador

Muitos navegadores mostram um aviso se recebem um certificado inválido. Navegadores mais antigos, quando se conectam a uma página com um certificado inválido, mostravam ao usuário um aviso em uma caixa de diálogo e perguntavam se ele desejava continuar. Navegadores mais recentes mostram o aviso preenchendo a janela inteira e também exibem as informações de segurança da página na barra de endereços. Certificados de validação estendida tornam verde a barra de endereço em navegadores mais recentes. A maioria dos navegadores exibe, também, um aviso ao usuário quando a página visitada contém uma mistura de conteúdo criptografado e não criptografado (uma página que utiliza HTTPS mas faz referência a links HTTP de alguma forma na página, por exemplo no link de uma foto).



A maioria dos navegadores, incluindo o Firefox (mostrado aqui), utiliza a barra de endereços para avisar ao usuário que sua conexão é segura, frequentemente colorindo o fundo.



A maioria dos navegadores, como essa imagem do Firefox, alerta o usuário quando uma página visitada possui um certificado de segurança inválido.

A Electronic Frontier Foundation opinou que "em um mundo ideal, toda requisição na web poderia utilizar HTTPS como padrão" e forneceu um complemento ao Firefox chamado "HTTPS Everywhere" (HTTPS em todo lugar) que funciona em várias páginas muito visitadas.

Aspectos técnicos

Diferenças para o HTTP

As URLs HTTPS começam com https:// e utilizam a porta 443 como padrão ou, alternativamente, 8443, enquanto as URLs HTTP começam com http:// e utilizam a porta 80 como padrão.
HTTP é inseguro e sujeito a ataques de homem-no-meio e escutas ilegais, que podem levar a atacantes ganharem acesso a informações sensíveis. O HTTPS foi projetado para proteger contra esses ataques e é considerado seguro contra eles (com exceção de versões mais antigas e obsoletas do SSL).

Camadas de rede

HTTP opera na camada superior do Modelo OSI, a camada de aplicação, mas o protocolo de segurança opera em uma subcamada inferior, criptografando uma mensagem HTTP antes de sua transmissão e decriptando a mensagem assim que ela chega ao destino. Estritamente falando, HTTPS não é um protocolo separado, mas se refere ao uso do HTTP sobre uma camada encriptada de conexão SSL/TLS.
Tudo na mensagem HTTPS é criptografado, incluindo os cabeçalhos, as requisições e respostas. Com a exceção de possíveis ataques criptográficos CCA descritos na seção de limitações abaixo, o atacante pode apenas conhecer o fato de que a conexão está sendo feita entre duas partes já conhecidas por ele, o nome do domínio e o endereço IP.

Configuração do Servidor

Para preparar uma página web de modo a aceitar conexões HTTPS, o administrador deve criar um certificado de chave pública para o servidor web. Esse certificado deve ser assinado por uma autoridade de certificação confiável para que o navegador aceite a conexão e não exiba avisos. A autoridade certifica que o proprietário do certificado é o operador do servidor web que o apresenta. Uma lista de autoridades de certificação de assinatura é geralmente distribuída aos navegadores web para que eles possam verificar certificados assinados por elas.

Adquirindo certificados

Certificados assinados por autoridades podem ser gratuitos ou custar entre 13 e 1.500 dólares por ano.
Organizações podem também ter sua própria autoridade de certificação, particularmente se são responsáveis por configurar navegadores para acessar suas próprias páginas (por exemplo, páginas de uma rede interna ou de uma grande universidade). Elas podem facilmente adicionar cópias de seus próprios certificados na lista de certificados distribuída no navegador.

Existe também uma autoridade de certificação ponto a ponto, a CACert.

Uso como controle de acesso

O sistema pode também ser utilizado para autenticação de clientes, com o objetivo de limitar o acesso a servidores web a usuários autorizados. Para fazê-lo, o administrador da página tipicamente cria um certificado para cada usuário, que é carregado em seu navegador. Normalmente, o certificado contém o nome e endereço de e-mail do usuário autorizado e é automaticamente verificado pelo servidor em cada reconexão para verificar a identidade do usuário, muitas vezes sem a necessidade de utilização de senhas.

Caso uma chave privada tenha sido comprometida

O certificado pode ser revogado antes de expirar, por exemplo por conta da chave privada ter sido comprometida. Versões mais recentes dos navegadores mais populares, como Google Chrome, Mozilla Firefox, Opera e Microsoft Internet Explorer, implementam o OCSP (Protocolo de Certificação Online de Estado) para verificar essa possível falha. O navegador envia, então, o número de série do certificado a uma autoridade de certificação via OCSP e a autoridade responde, informando ao navegador se o certificado ainda é válido.

Limitações

O protocolo TLS se apresenta em duas versões: simples e mútua. A versão mútua é mais segura, porém requer que o usuário instale um certificado pessoal em seu navegador para que possa se autenticar.

Independente da estratégia utilizada (simples ou mútua), o nível de proteção depende fortemente da corretude da implementação do navegador, da implementação do servidor e dos algoritmos criptográficos suportados.

O TLS não previne a página inteira de ser indexada utilizando um web crawler e, em alguns casos, a URI do recurso criptografado pode ser inferida sabendo-se apenas o tamanho da requisição ou da resposta. Isso permite a um atacante ter acesso ao texto plano (o conteúdo propriamente dito) e ao texto encriptado (a versão criptografada do texto plano), permitindo um ataque criptográfico.

Por conta do TLS operar abaixo do HTTP e não ter conhecimento de protocolos de níveis superiores, servidores TLS podem apenas apresentar um certificado para uma combinação particular de IP/porta. Isto significa que, na maioria dos casos, não é factível usar Hospedagem Virtual Baseada em Nome com HTTPS. Existe uma solução denominada Indicação de Nomes para Servidores (SNI), que envia o nome do host ao servidor antes de encriptar a conexão, embora muitos navegadores mais antigos não suportam essa extensão. Suporte para o SNI está disponível desde o Mozilla Firefox 2, Opera 8, Apple Safari 2.1, Google Chrome 6 e Microsoft Internet Explorer 7.Se controles parentais são ativados no Apple Mac OS X, páginas HTTPS devem ser explicitamente permitidas utilizando a lista Always Allow (Sempre Permitir).

De um ponto de vista arquitetural:

1. Uma conexão SSL/TLS é gerenciada pela máquina que inicializa a conexão TLS. Se. por algum motivo (roteamento, otimização de tráfego etc), essa máquina não é a aplicação do servidor e ela tem que decifrar dados, soluções têm de ser encontradas para propagar as informações de autenticação - ou certificado - do usuário para a aplicação do servidor, que necessita saber quem irá se conectar;
2. Para um TLS com autenticação mútua, a sessão SSL/TLS é gerenciada pelo primeiro servidor que inicializa a conexão. Em situações em que encriptação necessita ser propagada por servidores em cadeia, se torna mais difícil gerenciar o timeOut da sessão;

3. Com SSL/TLS com autenticação mútua, a segurança é maximal, mas no lado do cliente não há uma maneira de finalizar apropriadamente a conexão TLS e se desconectar. É preciso esperar que a sessão TLS expire ou finalize todas as aplicações do cliente;
4. Por motivos de desempenho, conteúdo estático que não é específico ao usuário ou à transação e, por isso, não privado, é normalmente entregue a um servidor não encriptado ou a uma outra instância sem TLS do servidor. Como consequência, esse conteúdo não fica protegido. Muitos navegadores alertam o usuário quando uma página misturou recursos criptografados e não criptografados.

Um ataque sofisticado de homem-no-meio foi apresentado na Blackhat Conference 2009. Esse tipo de ataque derrota a segurança fornecida pelo HTTPS modificando um link https: para um link http:, tirando proveito do fato de que poucos usuários da Internet digitam "https" nos seus navegadores. Os usuários normalmente alcançam páginas seguras clicando em um link, podendo ser, assim, levados a pensar que estão usando HTTPS quando de fato estão usando HTTP. O atacante, então, consegue se comunicar de modo não criptografado com o cliente.

Experiência do Usuário na navegação

Em julho de 2018 diversos navegadores incluindo Google Chrome (o mais utilizado no Brasil) começaram a deixar uma mensagem na barra do navegador para os Sites que não utilizam o Protocolo de segurança, colocando uma mensagem como "não seguro", e em páginas que utilizam dados dos usuários como e-mail, dados bancários e informações pessoais e não utilizam o HTTPS sofreram com a mensagem de forma explicita. A Google em sua plataforma de suporte e tutoriais aos desenvolvedores Web recomenda ações de segurança para melhorar a usabilidade e confiança dos usuários.

CORS

Cross-Origin Resource Sharing ou CORS é um mecanismo que permite que recursos restritos em uma página da web sejam recuperados por outro domínio fora do domínio ao qual pertence o recurso que será recuperado. Uma página da web pode integrar livremente recursos de diferentes origens, como imagens, folhas de estilo, scripts, iframes e vídeos. Certas "solicitações de domínio cruzado", em particular as solicitações Ajax, são proibidas por padrão pela política de segurança de mesma origem.

CORS define uma maneira pela qual um navegador e um servidor podem interagir para determinar se é ou não seguro permitir uma solicitação de origem cruzada. Isso permite mais liberdade e funcionalidade do que as solicitações da mesma origem, mas é mais seguro do que a simples autorização para ter todas as solicitações de origem cruzada. O padrão para CORS foi publicado originalmente como uma recomendação W3C, mas este documento está obsoleto. A especificação mantida ativamente que define o CORS é o Fetch Living Standard do WHATWG.

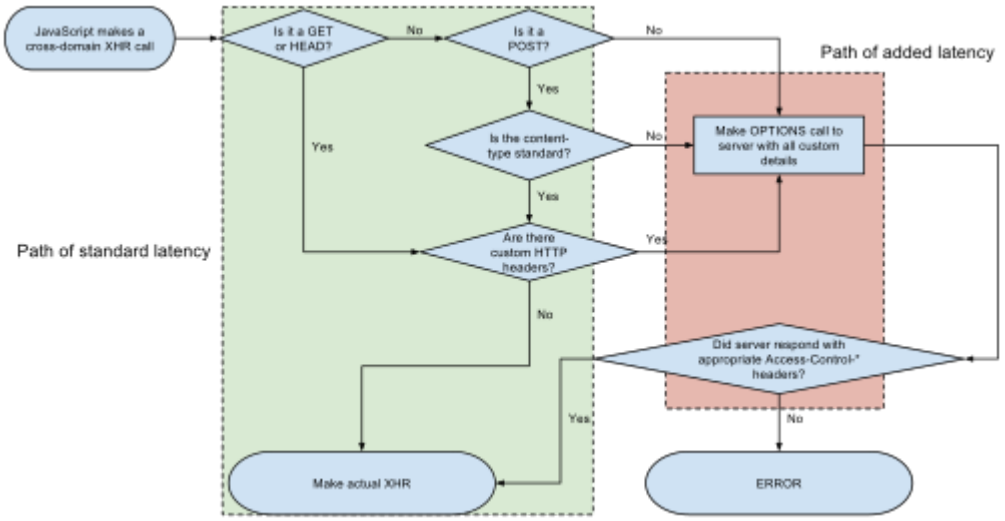
Histórico

O suporte de origem cruzada foi originalmente proposto por Matt Oshry, Brad Porter e Michael Bodell da Tellme Networks em março de 2004 para inclusão no VoiceXML 2.1, a fim de permitir consultas seguras de origem cruzada por navegadores VoiceXML. O mecanismo foi considerado de natureza geral e não específico para VoiceXML e foi subsequentemente separado em uma nota de implementação. O grupo de trabalho W3C WebApps, com a participação dos principais fornecedores de navegadores, começou a formalizar a nota em um rascunho de trabalho do W3C direcionado a uma recomendação formal do W3C.
Em maio de 2006, o primeiro esboço do projeto W3C foi apresentado. Em março de 2009, o projeto foi renomeado para

"Cross-Origin Resource sharing" e, em janeiro de 2014, foi aceito como uma recomendação W3C.

Como funciona o CORS

O padrão CORS descreve novos cabeçalhos HTTP que fornecem aos navegadores e servidores uma maneira de solicitar uma URL remota apenas se eles tiverem permissão. Embora certas validações e autorizações possam ser realizadas pelo servidor, geralmente é responsabilidade do navegador oferecer suporte a esses cabeçalhos e honrar as restrições que eles impõem.
Para métodos de solicitação Ajax e HTTP que podem modificar dados (geralmente métodos HTTP diferentes de GET, ou para usar POST com alguns tipos MIME), a especificação exige que os navegadores "verifiquem a solicitação upstream", solicitando os métodos suportados pelo servidor através de um método de pedido HTTP OPTIONS e, em seguida, após "aprovação" do servidor, enviar o pedido real com o método de pedido HTTP. Os servidores também podem notificar os clientes se "credenciais" (incluindo cookies e dados de autenticação HTTP) devem ser enviadas com a solicitação.



CORS e JSONP

O CORS pode ser usado como uma alternativa moderna ao modelo JSONP. Enquanto o JSONP oferece suporte apenas ao método de solicitação GET, o CORS também oferece suporte a outros tipos

de solicitações HTTP. O CORS permite que o programador da web use o **XMLHttpRequest** usual, que tem melhor tratamento de erros do que JSONP. Por outro lado, o JSONP funciona em navegadores mais antigos que são anteriores ao suporte ao CORS. CORS é compatível com a maioria dos navegadores da web modernos. Além disso, enquanto o JSONP pode causar problemas de segurança de cross-site scripting (XSS) quando o site está comprometido, o CORS permite que os sites processem manualmente as respostas para aumentar a segurança.

CSP (CONTENT SECURITY POLICY)

O que é CSP?

Basicamente é uma lista de domínios e regras (allowlist) que diz para o navegador de onde seu site vai poder carregar recursos como JavaScript, CSS, fontes, imagens, dentre outros.
Há duas maneiras de adicionar o CSP ao seu site:

- A primeira e mais recomendada é através dos headers HTTP.
- A segunda é através de meta tags no HTML, porém nem todas as regras de CSP funcionam dessa maneira e só é indicada quando não é possível utilizar os headers HTTP.

O CSP me protege do que?

Ele serve para proteger principalmente contra ataques XSS (Cross-site scripting), onde scripts ou conteúdos maliciosos são carregados em seu site sem permissão, alguns exemplos:

- keyloggers: quando o script faz um registro de tudo que é digitado pela vítima.
- cryptojacking: quando o script utiliza o computador dos usuários do seu site para minerar moedas digitais.

Um ataque antigo, mas que vale sua atenção, foi quando conseguiram injetar um script malicioso em um fórum sobre epilepsia, fazendo com que o site tivesse várias formas geométricas piscando cores diferentes, então pacientes

portadores da doença que acessaram o site no momento tiveram ataques de epilepsia.
Com isso em mente temos que considerar a evolução de tecnologias como o CSS, que com algumas linhas pode prejudicar os seus usuários, quebrar o layout ou tornar o seu site inacessível. Um outro exemplo são keyloggers utilizando apenas CSS.
Isso nos leva a um alerta ao pensar “inocentemente” que CSP é só sobre JavaScript, pois abre um mundo de possibilidades para pessoas má intencionadas.

O CSP funciona para todos os casos de XSS?

Não, esse é o caso do SELF XSS, que é quando o usuário envia comandos via console com permissões privilegiadas, é por isso que sites como o Facebook mantém esse tipo de mensagem em seu console:

Aviso de ESPERE no console de desenvolvedor no site do Facebook.

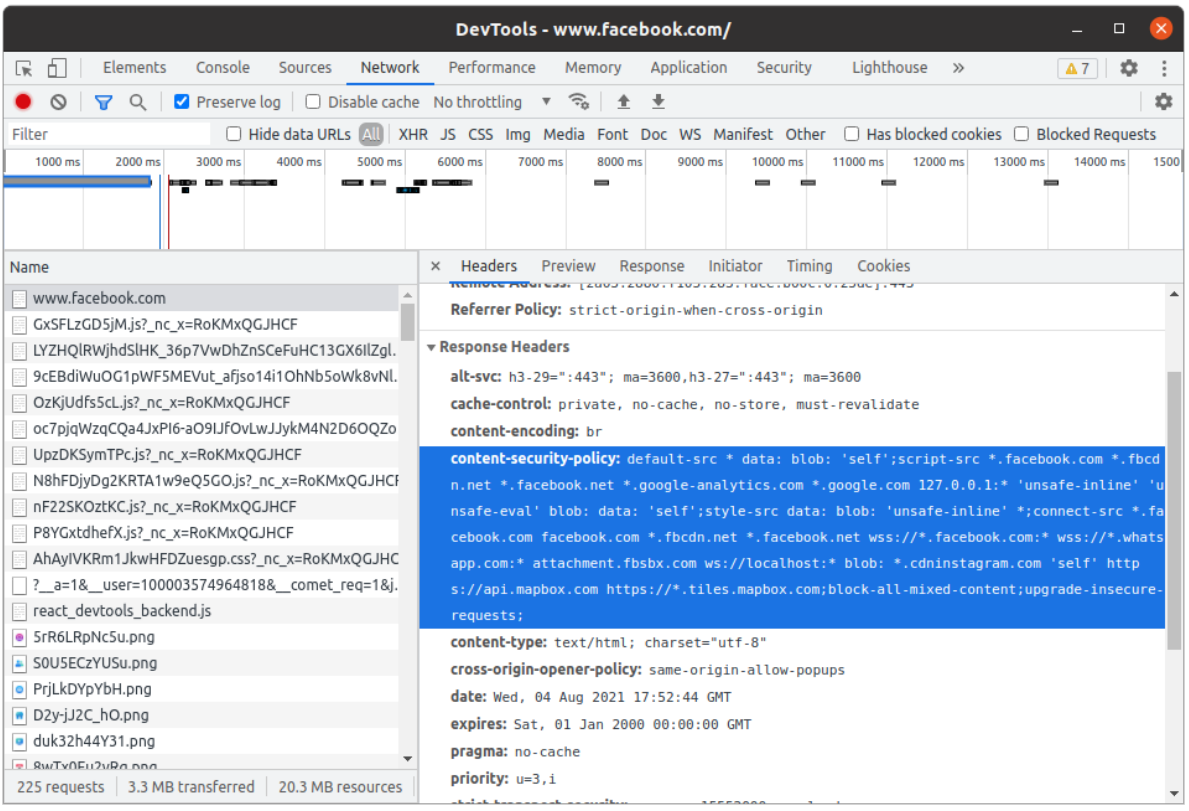
Ok, mas alguém usa isso? Eu nunca ouvi falar

O CSP é utilizado amplamente na web, porém ele não é simples de ser encontrado. Vou dar alguns exemplos de algumas empresas que utilizam:

1. Facebook
2. Instagram
3. Github
4. Apple

Para encontrar o CSP que foi adicionado nos headers HTTP, você deve:

1. Acessar o site desejado.
2. Abrir o devtools do navegador.
3. Ir na aba “Network”.
4. Pesquisar pela url que corresponde o domínio (ex: www.facebook.com).
5. Selecionar e abrir os headers, você deve ver algo como:



Facebook content-security-policy.

Agora se o CSP foi adicionado através de meta tags você pode encontrá-lo da seguinte forma:

1. Acessar o site desejado

2. Clicar com o botão direito do mouse na página e ir em ‘View Page Source’
3. Realizar uma pesquisa por ‘content-security-policy’, você deve ver algo como:

```
<meta http-equiv="Content-Security-Policy" content="default-src 'self'">
```

Como funciona?

O CSP utiliza como guia suas diretivas, são elas que ditam o tipo do conteúdo a ser carregado, como por exemplo:

- script-src: de onde os seus scripts JavaScript devem ser carregados
- style-src: de onde seus estilos devem ser carregados
- font-src: de onde suas fontes devem ser carregadas
- img-src: de onde suas imagens devem ser carregadas
- default-src: esta regra é muito útil, serve de fallback para quando você não tem uma regra específica para outra diretiva, exemplo: se você não declarar nenhuma das regras acima e declarar apenas o default ela servirá para as outras.

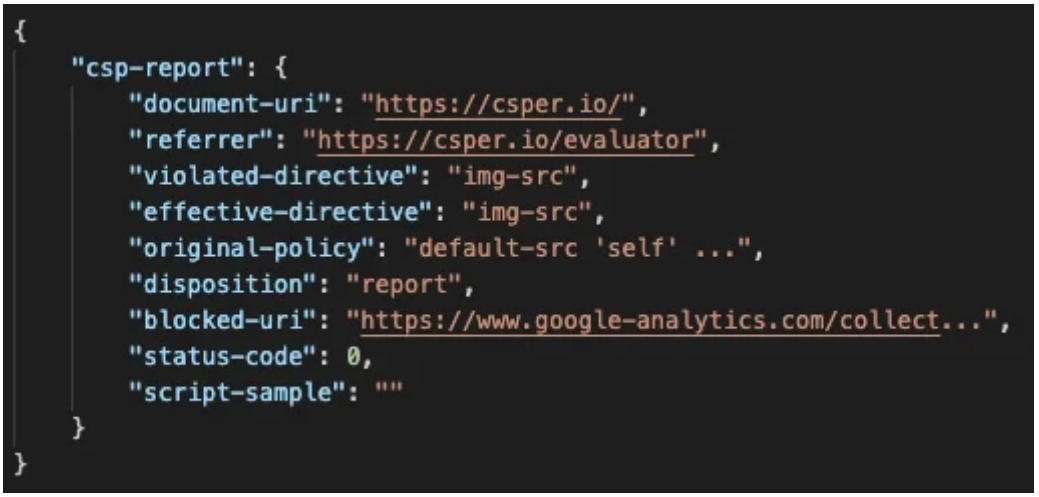
Porém cuidado com a utilização do default-src, ele cobre a maioria das diretivas mas não todas, estas são as diretivas que não são cobertas:

- base-uri
- form-action
- frame-ancestors
- report-uri
- sandbox

Quero usar, mas gostaria de testar as regras antes

Há o Content-Security-Policy-Report-Only onde as ações não serão bloqueadas. Ele pode ser utilizado em paralelo ao CSP normal e aliado ao report-uri como diretiva, que indica uma url para que seja enviado um payload com as informações da tentativa de burlar alguma regra, a partir dai você pode

analisar os reports e a medida que forem diminuindo, passa-los para o CSP normal.



Exemplo de report gerado através do report-uri.

Pontos de atenção ao escrever um CSP

É sempre bom prestar a atenção ao escrever algumas regras, para evitar algum tipo de efeito 'surpresinha 🎁', pois elas podem acabar se anulando e tornar o seu site vulnerável mesmo com CSP.

Utilização dos unsafe

- unsafe-inline: permite que inline scripts e inline styles sejam carregados em seu site. A utilização do unsafe-inline pode anular algumas regras, a própria documentação não recomenda a utilização, a menos que você tenha certeza do que está fazendo.
- unsafe-eval: permite que javascript seja interpretado a partir de uma string, isso pode ser um risco inclusive para JQuery.

Utilização do wildcard

- *: o wildcard isolado permite o recurso ser carregado de qualquer URL que não seja (data: blob: filesystem: schemes). Isso pode se tornar um risco ao ser utilizado no default-src. Uma boa maneira de se utilizar seria apenas para completar URL ex: '*.cloudfare.com'.

Divergências entre headers

Também há a possibilidade de ocorrer divergências entre as regras contidas nos headers, um exemplo seria a utilização do header Strict-Transport-Security sem a diretiva de upgrade-insecure-requests no CSP, pois enquanto o Strict-Transport-Security diz que as conexões ao seu site feita pelos usuários devem ser como HTTPS, a ausência da diretiva upgrade-insecure-requests permite com que seu site carregue recursos de fontes não seguras (HTTP).

Conclusão

Através desse overview sobre o que é CSP, do que protege seu site e como implementar, podemos afirmar que ele é indispensável se você quer manter a segurança do seu site, mas também pode se tornar um problema caso as regras não estejam bem estruturadas. Com esse artigo espero ter ajudado a esclarecer um pouco mais esse tema, para mais sugiro o site Content-Security-Policy.

OWASP (OPEN WEB APPLICATION SECURITY PROJECT)

O **OWASP** (Open Web Application Security Project), ou Projeto Aberto de Segurança em Aplicações Web, é uma comunidade online que cria e disponibiliza de forma gratuita artigos, metodologias, documentação, ferramentas e tecnologias no campo da segurança de aplicações web.

Todas as ferramentas, documentos, fóruns e capítulos do OWASP são grátis e abertos a todos os interessados em aperfeiçoar a segurança em aplicações. Promovemos a abordagem da segurança em aplicações como um problema de pessoas, processos e tecnologia, porque as abordagens mais eficazes em segurança de

aplicações requerem melhorias nestas áreas. A OWASP é um novo tipo de organização. O fato de ser livre de pressões comerciais permite fornecer informação de segurança de aplicações imparcial, prática e de custo eficiente.

A OWASP não é filiada a nenhuma empresa de tecnologia, apesar de apoiar o uso de tecnologia de segurança comercial. Da mesma forma que muitos projetos de software de código aberto, a OWASP produz vários tipos de materiais de maneira colaborativa e aberta.

História

OWASP foi iniciada em 9 de setembro de 2001 por Mark Curphey. Jeff Williams serviu como voluntário do final de 2003 até setembro de 2011. O atual presidente é Tobias Gondrom e o vice-presidente é Josh Sokol.

A Fundação OWASP é uma organização 501(c)(3) organização sem fins lucrativos (nos EUA), foi criada em 2004 e oferece suporte a infraestrutura e aos projetos da OWASP.

Desde 2011, a OWASP é também registrada como uma organização sem fins lucrativos na Bélgica, sob o nome de OWASP Europa VZW.

Owasp Projetos

Esta é uma das divisões mais populares da OWASP, uma vez que dá aos membros a oportunidade de testar teorias e idéias livremente com o aconselhamento profissional e apoio da comunidade OWASP. Você pode visualizar todas as listas, examinar seus arquivos, e subscrever qualquer projeto, visitando as Listas de Discussão do Owasp Projeto.

Inventário do OWASP Projetos

Flagship Projetos

A designação OWASP Flagship é dada aos projetos que tenham demonstrados valor estratégico para OWASP e segurança do aplicativo como um todo.

Projetos de Laboratório

Representam projetos que produziram um produto de valor, embora normalmente não estejam prontos para a produção.

Incubadora de Projetos

Representam os projetos que estão em desenvolvimento, ideias sendo comprovadas e projetos sendo concretizados.

Low Activity Project

Projetos de baixa atividade, projetos que não tiveram lançamento em pelo menos um ano, no entanto, tem mostrado ser ferramentas valiosas

Owasp TOP 10

O OWASP Top 10 é um documento de conscientização para a segurança das aplicações web. O OWASP Top 10 representa um amplo consenso sobre o que são as falhas de segurança de aplicativos web mais importantes. Os membros do projeto incluem uma variedade de especialistas em segurança de todo o mundo que compartilharam seus conhecimentos para produzir essa lista.

O OWASP Top 10 possui licença gratuita, contando também com diferentes traduções, onde se é dada por usuários voluntários.

O OWASP Top 10 de 2017 foi:

1. Injeção de Código
2. Quebra de Autenticação
3. Exposição de Dados Sensíveis
4. Entidades Externas de XML
5. Quebra de Controle de Acesso
6. Configuração Incorreta de Segurança
7. Cross-Site Scripting (XSS)
8. Deserialização Insegura
9. Utilização de Componentes com Vulnerabilidades Conhecidas

10. Log e Monitoramento Ineficientes

Top 10 Controles Preventivos

O OWASP Top 10 Controles Preventivos é uma lista de técnicas de segurança que devem ser incluídos em cada projeto de desenvolvimento de software. Eles são ordenados por ordem de importância, sendo o primeiro o mais importante:

1. verificar a segurança cedo e frequentemente;
2. parametrizar consultas;
3. codificar dados;
4. validar todas as entradas;
5. implementar controles de identidade e autenticação;
6. implementar controles de acesso;
7. proteger os dados;
8. implementar LOG e detecção de intrusão;
9. aproveitar as estruturas de segurança e bibliotecas;
10. manipulação de erros e exceções.

SSL/TLS

O Transport Layer Security (TLS), assim como o seu antecessor Secure Sockets Layer (SSL), é um protocolo de segurança projetado para fornecer segurança nas comunicações sobre uma rede de computadores. Várias versões do protocolo encontram amplo uso em aplicativos como navegação na web, email, mensagens instantâneas e voz sobre IP (VoIP). Os sites podem usar o TLS para proteger todas as comunicações entre seus servidores e navegadores web.

O protocolo TLS visa principalmente fornecer privacidade e integridade de dados entre dois ou mais aplicativos de computador que se comunicam. Quando protegidos por TLS, conexões entre um cliente (por exemplo, um navegador da Web) e um servidor (por exemplo, wikipedia.org) devem ter uma ou mais das seguintes propriedades:

- A conexão é privada (ou segura) porque a criptografia simétrica é usada para criptografar os dados transmitidos. As chaves para essa criptografia simétrica são geradas exclusivamente para cada conexão e são baseadas em um segredo compartilhado que foi negociado no início da sessão (veja § Handshake TLS). O servidor e o cliente negociam os detalhes de qual algoritmo de criptografia e chaves criptográficas usar antes que o primeiro byte de dados seja transmitido (ver § Algoritmo abaixo). A negociação de um segredo compartilhado é segura (o segredo negociado não está disponível para bisbilhoteiros e não pode ser obtido, mesmo por um invasor que se coloque no meio da conexão) e confiável (nenhum invasor pode modificar as comunicações durante a negociação sem ser detectado).
- A identidade das partes em comunicação pode ser autenticada usando criptografia de chave pública. Essa autenticação pode ser opcional, mas geralmente é necessária para pelo menos uma das partes (geralmente o servidor).
- A conexão é confiável porque cada mensagem transmitida inclui uma verificação de integridade de mensagem usando um código de autenticação de mensagem para evitar perda não detectada ou alteração dos dados durante a transmissão.

Além das propriedades acima, a configuração cuidadosa do TLS pode fornecer propriedades adicionais relacionadas à privacidade, como sigilo de encaminhamento, garantindo que qualquer divulgação futura de chaves de criptografia não possa ser usada para descriptografar as comunicações TLS registradas no passado.

O TLS suporta muitos métodos diferentes para trocar chaves, criptografar dados e autenticar a integridade da mensagem (consulte § Algoritmo abaixo). Como resultado, a configuração segura do TLS envolve muitos parâmetros configuráveis e nem todas as opções fornecem todas as propriedades relacionadas à

privacidade descritas na lista acima (consulte § Troca de chave (autenticação), § Segurança de codificação e § Tabelas de integridade de dados).

Tentativas foram feitas para subverter aspectos da segurança das comunicações que o TLS procura fornecer, e o protocolo foi revisado várias vezes para lidar com essas ameaças de segurança (ver § Segurança). Os desenvolvedores de navegadores da Web também revisaram seus produtos para se defenderem de potenciais pontos fracos de segurança depois que eles foram descobertos (veja o histórico de suporte a TLS / SSL dos navegadores da Web).

O protocolo TLS compreende duas camadas: o registro TLS e os protocolos de handshake TLS.

O TLS é um padrão proposto pela IETF (Internet Engineering Task Force), definido pela primeira vez em 1999, e a versão atual é o TLS 1.3 definido no RFC 8446 (agosto de 2018). O TLS baseia-se nas especificações SSL anteriores (1994, 1995, 1996) desenvolvidas pela Netscape Communications para adicionar o protocolo HTTPS ao navegador da Web Navigator.

Descrição

Aplicações cliente-servidor fazem uso do protocolo TLS para se comunicar através de uma rede de forma a prevenir a interceptação e adulteração da informação.
Uma vez que aplicações podem se comunicar tanto através de TLS (ou SSL) como sem ele, é necessário que o cliente sinalize ao servidor para a configuração de uma conexão TLS. Uma das maneiras de se obter isso é utilizar números de porta diferentes, por exemplo a porta 443 para HTTPS. Outro mecanismo é uma requisição específica por parte do cliente ao servidor para uma transição para a conexão TLS; por exemplo, ao fazer uma requisição STARTTLS ao utilizar protocolos de email.

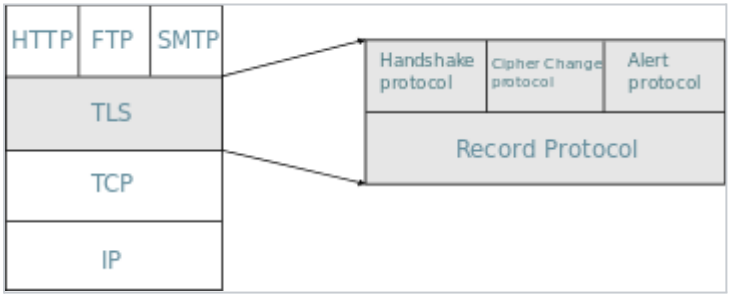
Uma vez que o cliente e o servidor concordaram quanto ao uso do TLS, eles negociam uma conexão de estado por meio de um procedimento de handshake. Os protocolos utilizam um handshake com uma chave pública para estabelecer as configurações de criptografia e uma chave de sessão única compartilhada através da qual toda a comunicação é criptografada utilizando uma chave simétrica. Durante esse handshake, o cliente e o servidor concordam a respeito dos vários parâmetros necessários para estabelecer a segurança da conexão:

- O handshake é iniciado quando o cliente se conecta a um servidor habilitado para TLS requisitando uma conexão segura e apresentando uma lista de algoritmos suportados (cifras e funções hash).
- A partir dessa lista, o servidor seleciona uma cifra e uma função hash para as quais também tenha suporte e notifica ao cliente a decisão.
- O servidor então geralmente apresenta informações de identificação na forma de um certificado digital. O certificado contém o nome do servidor, a autoridade de certificação (CA) que concedeu o certificado, e a chave pública do servidor.
- O cliente confirma a validade do certificado antes de continuar.
- Para gerar as chaves de sessão utilizadas na conexão segura, o cliente:
    - criptografa um número aleatório com a chave pública recebida e envia o resultado ao servidor (o único capaz de descriptografar a mensagem com sua chave privada); ambas as partes então fazem uso do número aleatório para gerar uma chave de sessão única para a criptografia subsequente dos dados durante a sessão, ou
    - inicia uma troca de chaves de Diffie-Hellman para gerar seguramente uma chave de sessão aleatória e única, utilizada para criptografar e descriptografar os dados e que ainda possui a propriedade de foward secrecy: caso a chave privada do servidor seja vazada no futuro, ela é incapaz de descriptografar a

sessão atual, mesmo que esta tenha sido interceptada e gravada por um terceiro.

Isso conclui o handshake e inicia a conexão segura, que é criptografada e descriptografada com a chave de sessão até o fim da conexão. Se qualquer um dos passos acima falhar, o handshake TLS também falha e a conexão não é criada.
Os protocolos TLS e SSL não se encaixam perfeitamente em nenhuma camada dos modelos OSI ou TCP/IP. O TLS é implementado "sobre um protocolo de comunicação confiável (por exemplo, o TCP)", o que implica que ele está acima da camada de transporte. Ele serve para criptografar as camadas superiores, o que normalmente seria função da camada de apresentação.

Contudo, as aplicações geralmente fazem uso do TLS como se fosse uma camada de transporte, mesmo que essas aplicações devam controlar ativamente o início dos procedimentos de handshake e o gerenciamento dos certificados de autenticação compartilhados.



Protocolo TLS

Funcionamento

O servidor do site que está sendo acessado envia uma chave pública ao browser, usada por este para enviar uma chave secreta simetrica, criada aleatoriamente. Desta forma, fica estabelecida a troca de dados criptografados entre dois computadores.Baseia-se no protocolo TCP da suíte TCP/IP e utiliza-se do conceito introduzido por Diffie-Hellman nos anos 70 (criptografia de chave pública) e Phil Zimmermann (criador do conceito PGP).

História e desenvolvimento

A primeira versão foi desenvolvida pela Netscape em 1994. O SSL versão 3.0 foi lançado em 1996, e serviu posteriormente de base para o desenvolvimento do TLS versão 1.0, um protocolo padronizado da IETF originalmente definido pelo RFC 2246. Grandes instituições financeiras como Visa, MasterCard, American Express, dentre outras, aprovaram o SSL para comércio eletrônico seguro na Internet.O SSL opera de forma modular, possui design extensível e apresenta compatibilidade entre pares com versões diferentes do mesmo.O SSL executa a autenticação das 2 partes envolvidas nas comunicações (cliente e servidor) baseando-se em certificados digitais.

Segurança

Ataques significativos contra TLS/SSL estão listados abaixo.
Em fevereiro de 2015, a IETF emitiu um informativo RFC resumindo os vários ataques conhecidos contra TLS/SSL.
A Apple corrigiu a vulnerabilidade BEAST implementando a divisão 1/n-1 e ativando-a por padrão no OS X Mavericks, lançado em 22 de outubro de 2013.

TESTES

TESTE DE INTEGRAÇÃO

O que é teste de integração?

Qualquer software consiste em diferentes módulos que executam diferentes funções. Um módulo de software pode funcionar bem de forma independente, mas para garantir que o software funcione bem quando todos os módulos são testadores integrados, realizam o Teste de Integração.
O teste de integração avalia a funcionalidade de vários

módulos quando integrados para formar uma única unidade. Este teste valida transições suaves entre vários componentes integrados do software. O objetivo do teste de integração é encontrar defeitos e falhas entre várias interfaces do software.

Por que precisamos de testes de integração?

A necessidade de teste de integração deve-se aos seguintes motivos:

• Os defeitos nos módulos integrados são difíceis de encontrar e complexos de corrigir.

• Corrigir bugs em módulos integrados pode eliminar o número geral de bugs e diminuir a contagem de alterações de código no próximo nível de teste do sistema.

• O teste de integração nos permite verificar a integração de vários componentes do software que ajudam a avaliar o ciclo de ponta a ponta do software.

• Essa abordagem de teste também reduz o risco de falha de software.

• Ele também valida o impacto das mudanças estruturais quando o usuário final muda de um módulo para outro.

• Isso ajuda a validar a funcionalidade correta das ferramentas API de terceiros no software.

• Esse teste aborda problemas que foram ignorados nos testes de unidade, como interceptação de erros, interfaces de hardware, formatação de dados e interfaces de terceiros. O teste de integração também verifica o comportamento funcional e não funcional das interfaces integradas.

Quais são os benefícios do teste de integração?

A abordagem de teste oferece uma lista de benefícios, como:

• Os testes de integração permitem garantir a adequação dos módulos e seus resultados.

• Também ajuda a detectar os problemas relacionados à interface entre os módulos.

• O teste de integração ajuda a estimular a interação entre vários módulos.
• Ele cobre vários módulos para fornecer uma cobertura de teste mais ampla.
• O teste de integração impacta a eficiência geral do teste do software.
• Isso ajuda a suavizar a transição entre as interfaces.

Como abordar o teste de integração?

Combinar várias unidades funcionais e testá-las para verificar os resultados é a abordagem utilizada no teste de integração.
O teste de integração tem duas subdivisões como abordagem incremental e big bang . O teste de integração incremental é dividido em três abordagens, de cima para baixo, de baixo para cima e Sandwich. Cada método tem seus benefícios.

Abordagem de teste incremental

Esta forma de teste de integração é realizada em módulos relacionados logicamente combinados em uma unidade. Todos os módulos são adicionados um a um na unidade de teste até que os testadores cubram todo o sistema. O teste incremental geralmente usa stubs e drivers, pois alguns módulos ainda estão em fase de desenvolvimento. Stubs e drivers são programas fictícios que não implementam a lógica, mas ajudam a estabelecer a comunicação.
Esta abordagem tem três abordagens para selecionar módulos:

1. **Teste de cima para baixo**

A abordagem descendente permite que os testadores testem o primeiro módulo superior isoladamente e, em seguida, adicionem outros módulos um por um à unidade de teste descendo. O fluxo de controle de teste no sistema vai de cima para baixo nesta abordagem. Como os testadores podem encontrar os módulos inferiores ainda em fase de desenvolvimento, eles usam stubs como programas fictícios que devolvem o controle aos módulos superiores.
Os benefícios desta abordagem são:

• Fácil localização de falhas
• Projeto de protótipo rápido
• Priorizando módulos críticos
• Fácil e rápido de encontrar falhas de design e consertar

A única desvantagem dessa abordagem é a falta de testes para módulos de nível inferior que estão em desenvolvimento.

2. Teste ascendente

Essa abordagem testa o módulo mais baixo primeiro na estrutura arquitetônica. Os outros módulos são adicionados de forma incremental um a um para testar o movimento ascendente na estrutura arquitetônica. O controle de fluxo no teste se move de baixo para cima.

Este método de teste é implementado quando os módulos superiores estão em construção. O método usa drivers para estimular o funcionamento dos módulos ausentes. Os drivers executam várias tarefas, como chamar o módulo em teste, passar dados de teste ou receber dados de saída.

Benefícios:

• Fácil de testar e desenvolver o produto
• Abordagem eficiente para teste de integração
• Fácil de criar condições de teste

3. Abordagem Sandwich

A abordagem Sandwich envolve testar os módulos inferior e superior na estrutura arquitetônica do software.

Nessa abordagem, a camada intermediária é o alvo que o testador deve alcançar. Todo o sistema de software é convertido em três camadas para iniciar o teste.

A primeira é a camada do meio, a segunda é a camada acima e a terceira é a camada abaixo. O objetivo de um testador é alcançar a camada intermediária testando ambas as camadas simultaneamente.

4. **Integração do big bang**

Essa abordagem de teste de integração é começar o teste quando todos os módulos forem desenvolvidos. Todas as unidades são testadas juntas como uma única unidade combinada. Essa forma de teste de integração funciona bem para sistemas menores. Existem certas limitações enfrentadas na integração do big bang, como:

• É um desafio encontrar erros de localização que atrasem o teste.

• Também é difícil encontrar a causa raiz de um defeito neste método.

• Todas as interfaces são testadas ao mesmo tempo, portanto, a equipe de teste deve investir mais esforço e tempo.

• Módulos críticos não são priorizados, o que poderia aumentar o risco geral.

Quais são as etapas para o teste de integração?

O teste de integração inclui as seguintes etapas:

1. O teste de integração começa com a compreensão da arquitetura do aplicativo.
2. Encontrando vários módulos do sistema.
3. Compreender a funcionalidade de cada módulo.
4. Avalie a transação de dados entre as interfaces.
5. Observe os pontos de entrada e saída no sistema.
6. Prepare o plano para o teste.
7. Selecione a abordagem de teste.
8. Categorize os módulos de acordo com as necessidades de teste.
9. Encontre as várias condições de teste para casos de teste.
10. Projete cenários de teste, scripts e casos de teste .
11. Implante os módulos escolhidos e execute o teste de integração.
12. Execute os casos de teste.
13. Rastreie os defeitos e registre os resultados.

Conclusão

O teste de integração é necessário para construir um sistema robusto que se comporte conforme o esperado. Este método de teste garante que as unidades individuais funcionem bem juntas.

TESTE UNITÁRIOS

Teste de unidade é toda a aplicação de teste nas assinaturas de entrada e saída de um sistema. Consiste em validar dados válidos e inválidos via I/O (entrada/saída) sendo aplicado por desenvolvedores ou analistas de teste.

Uma unidade é a menor parte testável de um programa de computador. Em programação procedural, uma unidade pode ser uma função individual ou um procedimento. Idealmente, cada teste de unidade é independente dos demais, o que possibilita ao programador testar cada módulo isoladamente.
Relação de conceitos de testes de unidade:
I/O Input Output (Entrada e Saída): são todas as entradas e saídas existentes na programação.

Válidos

São entradas e saídas de dados comuns ao sistema e pertencem ao processo normal. Não apresentam tratamento além do normal já programado. No caso de retorno deverá seguir os padrões estabelecidos e não permitir retornos fora das regras especificadas.

Inválidos

São entradas e saídas de dados não comuns ao sistema. Apresentam tratamento para validar o tipo de dado inválido ou situação. Pode apresentar até dois retornos, uma mensagem para

um log no sistema e uma mensagem com formatação e escrita adequada ao usuário.
Ex.: Dividir (x int,y int)=z int .
Caso tenhamos x=1 e y=0, z será um valor com erro e deverá retornar uma mensagem ao usuário, avisando que a operação é inválida. Caso a expressão seja um dado comum do sistema, a autorização para tal validação deverá ser do usuário, pois faz parte do conjunto de regras de negócio. Não existe retorno inválido sem um tratamento. O tratamento genérico será apenas para condições não visíveis na regra e uso do sistema.

Domínio

Pode ser um campo, uma assinatura, um I/O, ou qualquer tipo de local que receba valores externos ao sistema. Todo domínio deve realizar consistências de dados válidos e inválidos. Um domínio só permite dados com a formatação igual ao que será armazenado.
Ex.: Campo DDD deverá permitir números de até quatro casas não negativas ou a base de dados deve impedir a entrada de valores inválidos.

Receber e guardar o mesmo tipo de dado, o tamanho do campo que recebe os dados deve ser menor ou igual ao campo que irá armazenar os dados (em raros casos os campos de armazenamento são menores que os de exibição).
Em suma, domínio é o tipo de valor válido para cada campo. Como exemplo podemos citar: Campo nome: Dominio = tipo: string; tamanho:50. Ao aplicarmos o particionamento por equivalência e a análise por valor limite, poderemos criar as seguintes classes de testes.
Particionamento por Equivalência: campo nome:

- valor em branco; Cenário Negativo
- valor > 50; Cenário Negativo
- qualquer valor de 1 a 50; Cenário Positivo

Análise por Valor Limite:
campo nome: valor em branco; valores 49,50,51; Usamos um valor exatamente inferior e exatamente posterior ao valor do campo, devido ao fato dos erros aparecerem nas fronteiras da aplicação.

Tipos de valores

Existem diversos formatos de valores e cada um possui uma regra própria além das regras da região em que será usado.

Data

Deve validar anos iguais à faixa de datas existente na base. Deve ser feita conforme o padrão da região de uso. Validar o maior e o menor dia de todos os meses. Deve estar preparado para anos bissextos. Em caso de cálculo de feriados deve apresentar todos eles corretamente conforme as regiões definidas pelo usuário. Validar meses e dias invalidados conforme regra (X,X-1,X+1,(X+Y)/2,Y,Y-1,Y+1) sendo X o maior número e Y o menor. Verificar quantos anos o sistema deverá tratar anteriores e posteriores.

Números

Devem ser tratados no formato correto e com as regras para a região especificadas pelo usuário. Validar valores negativos ou sinais. Testar sempre o valor (um acima do maior, o maior, um abaixo do maior, alguns intermediários, um acima do menor, o menor, um abaixo do menor, e alguns valores inválidos) (X,X-1,X+1,(X+Y)/2,Y,Y-1,Y+1) sendo X o maior número e Y o menor número. Tratar valores fracionados conforme a quantidade de casas validadas pela empresa.

E-mail

Todo e-mail válido possui um "@" e um ".", sendo que o "@" nunca está no começo ou no fim, o "." deve estar sempre entre o "@" e o fim, o "." não pode estar junto com o "@", e antes

do "@" não pode haver caracteres especiais. Um e-mail dever ter no mínimo 5 caracteres contando o "@" e o ".". Ex.: nome@dominio.com

Senha

Deve atender as normas da empresa. Não deve ser exibida durante a digitação; somente com caractere representativo. Deve ser confirmada a digitação. Não deve permitir ser copiada ou colada.

Domínio

Existem diversas formas para validar domínio e e-mail válido, porém devem ser aprovadas pela empresa, pois consomem comunicação com outros sites.

Campos especiais CPF, CNPJ, CEP, CNS, etc.

- Validar a formatação na entrada dos dados e na armazenagem.
- Devem atender as regras de cálculo existentes.
- Não podem ser dados viciados.
- Campos dependentes DDI-DDD-TEL e similares.
- Validar o preenchimento de ambos em validação única.
- Validar a formatação.
- Campos pré-preenchidos.
- Validar a formatação.
- Testar todas as regras de preenchimento.
- Força dados inválidos ou repetir o preenchimento.

Valores para teste

Existem diversos tipos de dados válidos que se tornam inválidos conforme a linguagem usada.

Sintaxes da linguagem

Muitas linguagens podem ter problemas ao tratar valores do tipo objeto, string e outros. Tais linguagens têm tratamentos

especiais para estes valores para assim evitar erros ou em alguns casos não tem tratamento pois o erro é na codificação.

Erro na linguagem

Um exemplo claro de erro na linguagem são sistemas feitos na própria base de dados que não permitem pesquisas usando termos como else, while, for por serem palavras reservadas da linguagem. Atualmente a maior parte dos sistemas não apresenta erros com termos reservados.

Alteração de valores

Algumas linguagens, ao receberem determinados valores, alteram seus dados internos com tratamentos automáticos. No geral elas possuem travas para estes tratamentos. Exemplos de valores: 0x00, 00FF, 1.1, 1,1, 1^2, etc.

Interpretação de valores

Algumas linguagens interpretam diretamente valores sem tratar, passando eles para outra aplicação em uso. Este erro ocorre geralmente quando uma outra linguagem é usada como intermediária. Exemplo: em HTML (ASP, JSP, CGI, PHP e outras), quando passamos uma estrutura HTML(TAGs) para a página e esta estrutura é apresentada novamente, porém no corpo do HTML podemos ter dados incorporados que mudam o resultado apresentado. Este tipo de erro é chamado de inject. Outro exemplo de inject é o SQL em junção com outras linguagens, recebendo diretamente linhas de comando, podendo alterar pesquisas ou apagar tabelas de dados inteiras. Ex.: HTML(<TAGs>,javascript) SQL(términos de linhas seguidas de comandos (1;’ while 1=1 print 1)). Estes erros ocorrem pois temos uma segunda linguagem que faz o intermédio entre os comandos. Para tratar estes tipos de erros, devemos sempre conhecer bem as linguagens usadas, evitando transferir direto dados do usuário que acabem por virar comandos.

Cálculos mesclados

Ficar atento aos valores que são permitidos pois o usuário pode acabar por inserir dados inválidos. Excreções como (-,(),{},[],’,^,x,*,/,. e outras) podem mudar totalmente o valor que era esperado, podendo até colocar valores além dos permitidos pela especificação do sistema. Ponto flutuante ou Estouros. Erros de ponto flutuante dificilmente são identificados durante a especificação de um sistema ou acabam por ocorrer com a maturidade da empresa e seus dados. O que acarreta tais erros é: Não validar os dados corretamente. Não planejar bem o crescimento dos dados. Não ter uma área de armazenamento maior que a área de entrada de dados. Na época da inflação eram comuns sistemas terem um campo de valor de 9 casas e uma área de armazenamento com 12 casas. Outro erro comum que permite ocorrer estouro é uso de objeto e matrizes sem estudo, qualquer laço entre eles ou o próprio crescimento descontrolado.

Formatações de região ou não

É bom sempre forçar o sistema a usar uma região única caso não se tratem todos os valores com os quais ele pode trabalhar. O não tratamento da região pode tanto mudar o valor de um pagamento como alterar a data de uma cobrança. Caso o cliente não defina uma região devemos cobrar dele tal definição e devemos sempre estar atentos aos padrões já existentes, evitando misturar dados como DD/MM com MM/DD e 1.100 com 1,1.

Medidas

Um dado extremamente importante são as medidas. Elas devem ser sempre iguais no sistema inteiro ou serem tratadas para isso. É extremamente recomendado usar uma medida única no sistema evitando assim erros como mandar um foguete para um planeta e errarmos ele por 65.000km (caso da NASA de um sistema que misturava dados em polegadas e dados em metros). Os erros de medidas não se aplicam somente a fórmulas diferentes, mas também a tamanhos (ex.: 100cm=1m=0.001km).

Data

A formatação de data é o erro mais comum da maioria dos sistemas. Muitas vezes o sistema foi todo desenvolvido para dd/mm/yyyy, porém, ao ser implantado, descobre-se que a data da base é mm/dd/yyyy. Para evitar este tipo de erro deve ser sempre requisitado ao usuário o seguinte: 1°: qual a configuração do servidor em que será usado o sistema e qual seu idioma. 2°: qual a configuração dos clientes que usam o sistema e quais os seus idiomas. Com estas informações é possível validar para qual data devemos converter os dados ou qual data devemos desenvolver.

Navegação

Pode ser tanto a navegação sobre o sistema ou a navegação entre sistemas. Sendo que somente a navegação dentro do bloco é validada como teste de unidade. Os itens Interna ao Sistema e Entre Sistemas são testes de integração.

Local

Navegação entre os campos ou funcionalidades da tela. Deve sempre atender a ordem: esquerda para direita, de cima para baixo. Caso o usuário use outro padrão, ele deve ter especificado previamente, pois está fora dos padrões de mercado. A navegação local deve também validar os valores entre campos e seus resultados. Atentar para alteração de valores em campos diferentes do usado. Algumas regras podem forçar as alterações de campos secundários. Devem-se validar todas as interações que geram estas alterações com dados válidos e inválidos.

Interna ao sistema. Teste Integrado

Consiste na navegação entre as telas do sistema ou suas estruturas internas. Deve seguir o padrão definido pelo usuário. Devemos atentar para alterações entre telas e todas as suas iterações (válidas e inválidas). Devemos sempre listar

as funcionalidades de navegação em um objeto único. Ex.: Apontar o objeto que controla o botão (x) (o objeto do (alt+f4)) para o mesmo objeto que controla o fechar do menu. A navegação interna deve ser sempre bem mapeada e deve ser tratada por quem libera uma funcionalidade ou um responsável por integrar ela ao sistema. Muitos sistemas travam pois suas funções ficam congestionadas entre elas por não serem tratadas com sinalização e eventos de integração (como funções de fechar, funções de novo, funções de próxima e anterior). Esse tipo de teste também é conhecido como Teste de Instrumentação.

Entre sistemas. Teste Integrado

Quando temos a integração entre sistemas deve ser programada sempre a prioridade entre eles, quem tem prioridade de gravar ou apagar um dado. Deve ser validada a chamada entre eles para se evitar congestionamento. Atualização de dados. Retorno e saída de dados. Formatações de entrada e saída.

Dicas para teste de unidade

Sete regras,ou boas práticas, para implementação de testes de unidade:

1. Escreva o teste primeiro
2. Nunca inicie com um teste que será bem sucedido
3. Comece com valores nulos, ou algo que não funcione
4. Não fique com medo de fazer algo trivial para fazer o teste funcionar
5. Desacoplamento e testabilidade andam de mãos dadas
6. Utilize teste de mock
7. Divida o seu teste em 3 partes, Preparação, Ação e Verificação (em inglês: Arrange, Act and Assert)

Existem outras listas de boas práticas para implementar teste de unidade que podem ser encontradas na comunidade.