Resumo - Teste

2 de junho de 2023 18:00

PPT 1:

- **Computação Paralela vs. Computação Concorrente:**
  - **Paralela** -> Utiliza hardware paralelo para executar computações mais rapidamente. A principal contribuição é a eficiência.
    - Divisão em Subproblemas;
    - Utilização ótima do hardware paralelo.
  - **Concorrente** -> Pode ou não fazer múltiplas execuções ao mesmo tempo. Melhora a modularidade, responsividade ou a manutenibilidade.
    - Quando pode iniciar a execução;
    - Como se dá a troca de informação;
    - Como gerir acesso aos recursos partilhados.

PPT 2 e 3:

- **Programação Distribuída vs. Programação Paralela:**
  - **Distribuída**:
    - Clientes fazem solicitações para o processamento de funções individuais, e normalmente separadas, a servidores.
    - O servidor pode distribuir para diversos processadores as diversas solicitações que recebe.
    - Os sistemas envolvidos, geralmente, usam SOs diferentes.
  - **Paralela**:
    - Vários PCs/Processadores cooperam para processar um problema dividido em várias subtarefas.
    - Os computadores envolvidos, geralmente, usam o mesmo SO.
- **Arquiteturas de Sistemas Paralelos:**
  - **Arquitetura do Processador e Tendências Lógicas:**
    - Desenvolvimento Atual: Processadores Multicore.
    - Nº de Transístores: Medida, grosseira, da complexidade e desempenho de um processador.
    - Lei de Moore: O nº de transístores por chip dobra a cada 18 a 24 meses. É um observação válida há mais de 40 anos, feita pelo cofundador da Intel, Gordon-Moore.
    - **Razões para o Desempenho Não ser Muito Mais Melhorado:**
      1. Taxa de Relógio;
      2. Melhorias na Arquitetura.
  - **Taxonomia de Flynn das Arquiteturas Paralelas:**
    - A classificação de Flynn caracteriza PCs paralelos pela organização do controlo global e os fluxos de dados e de controlo resultantes.
    - **Classificações de Flynn:**
      - SISD - Instrução Única Dados Únicos;
      - MISD - Instruções Múltiplas Dados Únicos;
      - SIMD - Instrução Única Dados Múltiplos (Muito Usado no Passado e nas GPUs);
      - MIMD - Instruções Múltiplas Dados Múltiplos (Muito Usado no Presente).
  - **Organização da Memória nos Computadores Paralelos:**
    - Os processadores ligam-se à memória global pela rede (SMM - Shared Memory Machine).
    - **Cache** -> Memória pequena, mas rápida, entre o processador e a memória principal.
    - **Processadores Multicore** -> Cada processador deve ter uma visão coerente do sistema de memória, ou seja, qualquer acesso de leitura retorna o valor escrito mais recente.
  - **Paralelismo a Nível de Thread:**
    - **Hyperthreading** -> É baseado numa duplicação da região do processador para o armazenamento do estado do processador no chip dos processadores.
    - **Resultado** -> O processador físico é visto como dois processadores lógicos (os processadores lógicos compartilham quase todos os recursos do processador físico).
    - **Opções de desenho de projeto para Processadores Multicore:**
      - Projetos hierárquicos, de Pipelines ou Baseados em Rede.
  - **Redes de Interconexão:**
    - **Redes de Comunicação de Computadores Paralelos:**
      - Usadas para comunicação em sistemas multi-computadores.
      - **Topologia:**
        - **Redes Estáticas/Diretas:**
          - Os nós são interconectados ponto a ponto.
          - Diâmetro Pequeno -> Distâncias pequenas de transmissão de mensagens.
          - Grau Pequeno de um Nó -> Reduz os requisitos de hardware.
          - Alta Bisseção de Banda Larga -> Ajuda a alcançar alto rendimento de dados.
          - Alta Conectividade -> Aumenta a fiabilidade.
        - **Redes Dinâmicas/Indiretas:**
          - Os nós são conectados indiretamente por vários switches intermediários.

                - ◇ Consistem em vários fios físicos e interruptores,
                - ◇ Os switches são configurados dinamicamente de acordo com os requisitos de transmissão da mensagem.
            - □ **Técnicas de Encaminhamento:**
                - ◆ **Algoritmo de Encaminhamento:**
                    - ◇ Seleção do caminho.
                - ◆ **Estratégia de Comutação:**
                    - ◇ Modo de transmissão.
    - **Encaminhamento e Switching (Chaveamento):**
        - **Algoritmos de Roteamento:**
            - □ Encontra um caminho na rede pelo qual uma mensagem do remetente A para o destinatário B deve ser enviada, segundo certas condições.
        - Uma **estratégia de switching** define como é que uma mensagem é enviada sobre um caminho escolhido pelo algoritmo de roteamento, desde o remetente até o destinatário.
    - **Caches e Hierarquia de Memória:**
        - Localidade de Acesso à Memória:
            - □ **Espacial:**
                - ◆ Acesso vizinho aos locais de memória principal em pontos consecutivos durante a execução do programa.
            - □ **Temporal:**
                - ◆ O mesmo local de memória é acedido várias vezes em pontos consecutivos durante a execução do programa.

PPT 4 e 5:

- **Modelos para Sistemas Paralelos:**
    - **Modelos de Máquinas Paralelas:** O nível mais baixo de abstração, descrição do hardware do sistema;
    - **Modelos de Arquitetura Paralelas:** Abstração dos modelos de máquinas, SIMD ou MIMD, organização de memória;
    - **Modelos Computacionais Paralelos:** Extensão dos modelos arquiteturais que permitem construir algoritmos para os custos serem considerados;
    - **Modelos de Programação Paralela:** Descrição de um sistema pela descrição da linguagem de programação e seu ambiente.
- **Paralelização de Programas:**
    1. **Decomposição das Computações:**
        - Decomposição do algoritmo em tasks;
        - Especificação das dependências das tasks;
        - As tasks têm acesso a variáveis partilhadas ou trocam mensagens usando operações de comunicação;
        - Granularidade de uma task: Uso relativo de operações na CPU vs. Operações de I/O.
    2. **Atribuição das Tarefas a Processos:**
        - Tem o propósito de executar um número semelhante de computadores por cada processo, tal que se obtenha uma boa distribuição da carga;
        - A atribuição das tarefas aos processos chama-se **escalonamento**.
    3. **Orquestração e Mapeamento dos Processos aos Cores e/ou aos Processadores Físicos:**
        - O propósito é minimizar os custos de comunicação e outros overheads associados.
- **Níveis de Paralelismo:**
    - Paralelismo de Instruções;
    - Paralelismo de Dados;
    - **Ciclos (Loops) Paralelos:**
        - **Há, entre outros, 2 modelos:**
            - □ **Loop Distribuído:** As instruções sem interdependência são separadas e podem ser paralelizadas.
            - □ **Paralelismo DOALL:** Não há dependências e as instruções de um loop podem ser executadas paralelamente.
    - Paralelismo de Funções;
    - **Padrões de Programação Paralela:**
        - **Padrões de Desenho Paralelo:**
            - □ **Estruturas de Coordenação das Threads:**
                - ◆ **Criação de Threads/Processos:**
                    - ◇ **Estática:** Um nº fixo de threads ou processos é criado no início da execução e são destruídas no final da execução.
                    - ◇ **Dinâmica:** A criação de threads ou processos é feita sempre que necessário.
                - ◆ **Fork-Join:**
                    - ◇ Uma thread existente um processo cria uma 2ª thread ou processo;
                    - ◇ O encadeamento é arbitrário;
                    - ◇ Diferentes linguagens e ambientes de programação exibem características diversas.
                - ◆ **Parbegin-Parend:**
                    - ◇ Criação e destruição simultânea de várias threads;
                    - ◇ As instruções no bloco são mapeadas para threads diferentes.
                - ◆ **SPMD e SIMD:**
                    - ◇ Todas as threads executam o mesmo programa com dados diferentes;
                    - ◇ **SIMD:** A mesma instrução é executada simultaneamente por todas as threads (síncrono).

        - ◇ **SPMD:** Num certo momento, threads diferentes executam instruções diferentes (assíncrono).
    - **Master-Slave ou Master-Worker:**
        - ◇ Uma única thread controla todas as computações de um programa;
        - ◇ Cria threads slaves ou workers e atribui-lhes computações.
    - **Modelo Cliente-Servidor:**
        - ◇ Várias threads clientes fazem solicitações à thread servidor, a qual atende os clientes de forma paralela;
        - ◇ **Extensões:** Várias threads servidor ou threads que são clientes e servidores ao mesmo tempo.
    - **Pipelining:**
        - ◇ Permite paralelismo, apesar das dependências dos dados;
        - ◇ Por exemplo, passar vários vídeos por uma série de filtros de software.
    - **Pool de Tasks:**
        - ◇ Uma estrutura de dados faz a gestão de partes do programa na forma de tasks;
        - ◇ A execução é por um nº fixo de threads, que vão acedendo ao pool para extração e armazenamento de taks.
    - **Threads Produtor-Consumidor:**
        - ◇ As threads produtoras criam os dados e as threads de consumo usam os dados;
        - ◇ Há uma estrutura de dados comum para o armazenamento dos dados.
- **Distribuição de Dados para Vetores e Matrizes:**
    - Os dados são particionados em conjuntos menores, sendo distribuídos aos cores ou processadores para usar na execução.
    - **Memória Distribuída:** Os dados atribuídos a um processador ficam armazenados na memória local e só podem ser acedidos por este processador.
    - **Memória Partilhada:** Os dados são armazenados na mesma memória partilhada e cada core ou processador acede a dados diferentes de acordo com o padrão de distribuição dos dados, que tem de evitar conflitos de acessos.
- **Troca de Informações:**
    - **Memória Distribuída:**
        - Operações de comunicação que permitem a troca de informação pelo envio de mensagens;
        - Distinção entre comunicação ponto-a-ponto e comunicação global.
    - **Memória Partilhada:**
        - **Uso de variáveis partilhadas cujo acesso concorrente é protegido por operações de sincronização:**
            - □ Sequencialmente dos acessos concorrentes;
            - □ Prevenção de condições de corrida usando locks.
- **Multiplicação Paralela Matriz-Vetor:**
    - **As duas representações sequenciais permitem duas implementações diferentes num hardware com P cores ou processos:**
        A. Representação da matriz A orientada a linhas e a computação de N produtos escalares;
        B. Representação da matriz A orientada a colunas e computação de uma combinação linear.
    - **Na arquitetura de memória partilhada, o modelo usado é o SPMD e não há distribuição explícita de dados:**
        - Cada core acede a uma parte diferente da matriz A;
        - Cada unidade de processamento computa N/P componentes do vetor resultante C e usa as correspondentes N/P linhas de A.
        - Não há conflitos de acesso.
- **Processos e Threads:**
    - **Processos:**
        - Programa em execução, com o código e toda a informação relacionada.
        - Tem um espaço próprio de endereçamento.
        - Faz mudanças de estado.
        - A criação e a gestão de processos tem um overhead relativamente elevado.
    - **Threads:**
        - É uma extensão do modelo de processo, onde um processo pode incluir vários fluxos de computação independentes.
        - Partilham o espaço de endereçamento do seu processo.
        - A criação de threads é relativamente rápida.
    - **Métodos de Sincronização:**
        - **Sincronização é a coordenação dos threads:**
            - □ Ordem desejada de execução;
            - □ Acesso à memória partilhada.
    - **Problemas e Soluções:**
        - **Correção em Computação Paralela:**
            - □ **Critério 1:** O output deve ser sempre o mesmo;
            - □ **Critério 2:** O output deve ser o mesmo se o código não for executado em paralelo.
        - **Condições de Corrida;**
        - **Estado Partilhado:**
            - □ **Problema do Banco:**
                - ◆ O estado partilhado entre múltiplos processos ou threads cria problemas que um ambiente single-thread não tem.
        - **Lock e Semáforos:**
            - □ Variáveis partilhadas podem ser usadas como sinais que todas as execuções entendem e respeitam, coordenando-se através da passagem de mensagens.

        - **Locks:** São também conhecidos como mutexes e são objetos partilhados usados para sinalizar que um estado partilhado está a ser lido ou modificado.
        - **Semáforos:** São sinais usados para proteger o acesso a recursos limitados: são semelhantes aos locks, mas podem ser adquiridos múltiplas vezes até um limite.
    - **Sincronização por Barreiras;**
    - **Variáveis de Condição:**
        - São úteis quando uma computação em paralelo é composta por uma série de passos e indicam quando uma condição foi satisfeita.
        - Um processo pode usar uma variável de condição para sinalizar que terminou a sua vez, o os outros podem começar a trabalhar.
    - **Deadlocks:**
        - Embora mecanismos de sincronização sejam eficientes para proteger o estado partilhado, podem causar que processos fiquem a esperar-se mutuamente.
        - É o **deadlock, uma situação em que 2 ou mais processos ficam bloqueados**, cada um à espera que o outro termine, mesmo se houver o nº correto de comandos acquire() e release().

PPT 4 e 5 - Complementar:

- **Modelos de Programação Paralela:**
    - **Memória Partilhada:**
        - A aplicação é uma coleção de threads que podem ser criadas dinamicamente.
        - Cada thread possui variáveis privadas e partilhadas.
    - **Memória Distribuída:**
        - A aplicação consiste numa coleção de processos diferentes, possivelmente em máquinas diferentes, configurados no arranque do programa e em que os dados não são partilhados, mas sim particionados pelos processos.
        - Os processos comunicam-se por mensagens e a coordenação está implícita nos eventos da comunicação.
    - **Dados Paralelos:**
        - Um único thread de operações paralelas;
        - Não se adapta a todos os problemas que podem ser paralelizados.
    - **Híbridos:**
        - MPI no topo da hierarquia, fazendo comunicação entre máquinas;
        - Memória partilhada pelos threads em cada máquina;
        - Modelo usado pelos supercomputadores.
- **Programação em Memória Partilhada:**
    - **Programação com Threads:**
        - Padrão POSIX portável mas pesado e lento.
        - **Variáveis Partilhadas -> Globais**
        - **Variáveis Privadas -> Locais**
    - **Programação com OpenMP:**
        - Suporta a programação científica.
        - É usado como alternativa à programação com threads.
        - É uma especificação aberta para processamento paralelo.
        - **Variáveis Partilhadas -> Shared**
        - **Variáveis Privadas -> Private**
        - **Permite:**
            - Separar o programa em regiões sequenciais e paralelas;
            - Confiar na gestão automática das necessidades de memória;
            - Usar mecanismos de sincronização já disponíveis.
        - **Não Permite:**
            - Paralelismo automático;
            - Garantias de melhor desempenho;
            - Evitar por completo inconsistências no acesso partilhado aos dados.
        - **Sincronização:**
            - **Secções Críticas:** #pragma omp critical
            - **Diretivas Barrier:** #pragma omp barrier
            - **Funções de Lock Explícitas:** omp_set_lock(lock1); /*código*/ omp_unset_lock(lock1);
            - **Regiões de Thread Único dentro de Regiões Paralelas:** #pragma omp single
- **Programação em Memória Distribuída:**
    - **Programação com OpenMPI:**
        - Não há variáveis partilhadas.
        - **Os programas executam semelhante a programas sequenciais uniprocesso, exceto pelas chamas à biblioteca de passagem de mensagens:**
            - **Comunicação** de dados;
            - **Sincronização** -> Não há locks pois não há variáveis partilhadas;
            - **Coordenação**.
        - **Coordenação com o Ambiente:**
            - **MPI_Comm_size** -> Informa o nº de processos a correr;
            - **MPI_Comm_rank** -> Informa o rank, um inteiro que identifica um processo;

        - **MPI_Comm_WORLD** -> Designa todos os processos em execução.
    - **Conceitos MPI:**
        - Os processos podem ser juntados em **grupos**;
        - Cada mensagem é enviada num **contexto**;
        - Um grupo e um contexto formam um **comunicador**;
        - Um processo é definido pelo seu **rank**;
        - O **MPI_Comm_WORLD** é um comunicador com todos os processos iniciais;
        - **Tag** é o identificador numérico de uma mensagem enviada.

PPT 5:

- **Uma aplicação é uma coleção de threads**, que podem, ou não, ser criadas dinamicamente.
- Cada thread possui variáveis privadas e partilhadas.
- **Programação em Memória Partilhada:**
    - **PTHREADS:**
        - Padrão POSIX, portável mas pesado.
        - **Variáveis Partilhadas -> Globais**
        - **Variáveis Privadas -> Locais**
    - **OPENMP:**
        - Suporta a programação científica.
        - **Variáveis Partilhadas -> Shared**
        - **Variáveis Privadas -> Private**
        - **Permite:**
            - Separar o programa em regiões sequenciais e paralelas;
            - Confiar na gestão automática das necessidades de memória;
            - Usar mecanismos de sincronização já disponíveis.
        - **Não Permite:**
            - Paralelismo automático;
            - Garantias de melhor desempenho;
            - Evitar por completo inconsistências no acesso partilhado aos dados.
        - **Sincronização:**
            - **Secções Críticas:** #pragma omp critical
            - **Diretivas Barrier:** #pragma omp barrier
            - **Funções de Lock Explícitas:** omp_set_lock(lock1); /*código*/ omp_unset_lock(lock1);
            - **Regiões de Thread Único dentro de Regiões Paralelas:** #pragma omp single
    - TBB:
        - Biblioteca de modelos C++ para programação paralela da Intel.
    - CILK e CILK++:
        - Linguagens para programação paralela baseadas em C e C++, com estrutura de threads menos pesada que PTHREADS.
    - **Threads em Java:**
        - Objetos baseados nos threads POSIX.

PPT 6:

- **Estilos de Arquitetura:**
    - **Ideia Básica:**
        - **Um estilo é formulado em termos de:**
            - Componentes com interfaces definidas;
            - A forma como os componentes estão ligados uns aos outros;
            - Os dados trocados entre componentes;
            - Como estes componentes e conectores são configurados conjuntamente num sistema.
    - **Conector:**
        - Mecanismo que media a comunicação, a coordenação ou a cooperação entre os componentes.
    - **Estilos:**
        - **Arquitetura em Camadas:**
            - **Camada de Interface de Aplicação:** Contém unidades para interagir com utilizadores ou aplicações externas;
            - **Camada de Processamento:** Contém as funções de uma aplicação, ou seja, sem dados específicos;
            - **Camada de Dados:** Contém os dados que um cliente quer manipular através dos componentes da aplicação.
        - **Arquitetura Baseada em Objetos e Serviços:**
            - **Essência:**
                - Os componentes são objetos, ligados entre si através de chamadas de procedimentos.
                - Os objetos podem ser colocados em diferentes máquinas e as chamadas pode, assim, ser executadas através de uma rede.
            - **Encapsulação:**
                - Diz-se que os objetos encapsulam dados e oferecem métodos sobre estes dados, sem revelar a implementação interna.
        - **Arquitetura Baseada em Recursos:**
            - **Arquiteturas RESTful:**
                - **Essência**: O sistema distribuído é uma coleção de recursos, e cada recurso é gerido por componentes. Os recursos podem ser adicionados, removidos, recuperados e modificados por aplicações.

        - **Operações Básicas**: PUT, GET, DELETE e POST.
    - **SOAP vs. RESTful:**
        - Muitos preferem o RESTful por a interface de um serviço ser mais simples.
        - O lado menos positivo é que muito precisa de ser feito no espaço dos parâmetros.
- **Arquitetura Editor-Assinante:**
    - Acoplamento temporal e referencial.
    -

	Temporalmente Acoplado	**Temporalmente Desacoplado**
Referencialmente Acoplado	Ligação Direta	Caixa de Correio
Referencialmente Desacoplado	Baseada em Eventos	Espaço de Dados Partilhado

- **Organização do Middleware:**
    - **Utilização de Entidades Legadas Para Construir Middleware:**
        - **Problema:**
            - As interfaces oferecidas por um componente legado provavelmente não são adequadas para todas as aplicações.
        - **Solução:**
            - Um Wrapper ou Adaptador oferece uma interface aceitável para uma aplicação do cliente.
    - **Organização dos Wrappers:**
        - **Duas soluções:**
            - **1-para-1**: Requer N * (N - 1) = $O(N^2)$ Adaptadores.
            - **Através de um broker**: Requer 2N = O(N) Adaptadores.
    - **Desenvolvimento de Middleware Adaptável:**
        - **Problema:**
            - O conceito de middleware implica em soluções que são boas para a maior parte das aplicações => pode-se desejar adaptar o seu comportamento para aplicações específicas.
- **Organizações Centralizadas:**
    - **Modelo Básico Cliente-Servidor:**
        - **Servidor** -> Processo que oferece serviços.
        - **Cliente** -> Processo que usa serviços.
        - Clientes e Servidores podem ou não estar na mesma máquina.
        - Clientes seguem o modelo solicitação/resposta para usar os serviços.
    - **Arquiteturas de Sistema Multiníveis:**
        - **Nível Único**: Configuração de terminal/computador central;
        - **Dois Níveis**: Configuração cliente/servidor único;
        - **Três Níveis**: Cada camada numa máquina separada.
- **Arquiteturas Descentralizadas:**
    - **Organizações Alternativas:**
        - **Distribuição Vertical:**
            - Vem da divisão de aplicações distribuídas em 3 níveis lógicos, e da execução dos componentes de cada nível num servidor diferente.
        - **Distribuição Horizontal:**
            - Um cliente ou servidor pode estar fisicamente dividido em partes logicamente equivalente, mas cada parte está a operar na sua própria parte do conjunto de dados completo.
        - **Arquiteturas P2P:**
            - Os processos são todos iguais: as funções que precisam de ser executadas são representadas por todos os processos; cada processo funcionará como cliente e servidor ao mesmo tempo.
    - **P2P Estruturado:**
        - **Essência:**
            - Em redes estruturadas P2P a estrutura lógica é organizada numa topologia específica, e o protocolo garante que qualquer nó pode pesquisar eficientemente a rede por um ficheiro/recurso, mesmo que o recurso seja raro.
    - **P2P Não Estruturado:**
        - **Essência:**
            - Cada nó mantém uma lista ad hoc de vizinhos. A estrutura lógica resultante assemelha-se a um grafo aleatório: uma aresta existe somente com uma certa probabilidade.
        - **Pesquisa na Rede P2P Não Estruturada:**
            - **Inundação (flooding):**
                - O nó emissor passa o pedido pelo recurso a todos os vizinhos. O pedido é ignorado quando o nó recetor já o tinha visto antes. Caso contrário, procura localmente (recursivamente).
            - **Caminhada Aleatória (random walk):**
                - O nó emissor passa o pedido pelo recurso a um vizinho escolhido aleatoriamente. Se esse vizinho não tiver o recurso, ele reencaminha o pedido a um dos seus vizinhos escolhido aleatoriamente, e assim continua.
        - **Inundação vs. Caminhada Aleatória:**
            - A caminhada aleatória é mais eficiente na comunicação, mas pode demorar mais tempo até encontrar o recurso.
- **Modelos Híbridos:**
    - **Modelos Híbridos Para Redes P2P:**
        - **Essência:**
            - Os modelos híbridos são uma combinação de modelos P2P e cliente-servidor.
            - Um modelo híbrido comum é ter um servidor central que ajuda os pares a encontrarem-se.

        - Existe uma variedade de modelos híbridos, com diversas relações custo/benefício entre a funcionalidade centralizada fornecida por uma rede estruturada de servidor/cliente e a igualdade de nó proporcionada pelas redes puramente P2P não estruturadas.
    - **Modelo Skype: A deseja contatar B:**
        1. **Tanto A quanto B estão na Internet Pública:**
            - Uma ligação TCP é estabelecida entre A e B para pacotes de controlo.
            - A chamada real dá-se utilizando pacotes UDP entre portas negociadas.
        2. **A Opera Atrás de um Firewall, e B Está na Internet Pública:**
            - A estabelece uma ligação TCP para pacotes de controlo com um servidor S.
            - S estabelece uma ligação TCP com B para repassar pacotes de controlo.
            - A chamada real dá-se utilizando pacotes UDP e diretamente entre A e B.
        3. **Tanto A como B Operam Atrás de Firewalls:**
            - A conecta-se com um servidor S utilizando TCP.
            - S estabelece uma ligação TCP com B.
            - Para a chamada real, outro servidor R é contatado como estafeta (relay): A estabelece uma ligação com R, r B também o faz.
            - Todo o tráfego de voz é direcionado sobre as duas ligações TCP, e através do servidor R.
    - **Arquitetura de Servidores na Periferia (Edge Computing):**
        - **Essência:**
            - A edge computing é aquela na qual o processamento acontece no local físico do cliente, ou da fonte de dados.
            - Com o processamento mais próximo, os utilizadores beneficiam-se de serviços mais rápidos e fiáveis, e as empresas desfrutam da flexibilidade de usar e distribuir um conjunto de recursos por um grande nº de locais.

PPT 7:

- **Socket:**
    - End Point da comunicação.
    - Envio de mensagens entre processos via rede.
    - O processo envia e recebe via um socket - a porta da cada que identifica a aplicação.
    - Socket é um API que dá suporte à criação de aplicações de rede.
    - **Socket de Datagrama (UDP):**
        - Coleção de mensagens;
        - Sem garantias, melhor esforço;
        - Sem estabelecimento de sessão.
    - **Socket de Stream (TCP):**
        - Fluxo de bytes;
        - Garantias de entrega e integridade;
        - Estabelece uma sessão subjacente ao tráfego de dados.
    - **Caracterização de um Socket:**
        - Protocolo de Comunicação (TCP ou UDP);
        - Endereço IP (para identificar o hospedeiro);
        - Número da Porta (identifica o processo e tem 16 bits).
- **Clientes e Servidores:**
    - **Programa Cliente:**
        - Inicia a comunicação;
        - Tem de saber o IP e porta do servidor;
        - Solicita um serviço.
    - **Programa Servidor:**
        - Aguarda ligações;
        - Obtém o IP e porta do cliente na ligação;
        - Fornece o serviço.
- **Ambiente Tradicional de RPC vs. Java RMI:**
    - **RPC: Remote Procedure Call:**
        - Chamada provedoral de um processo de uma máquina servidora a partir de um processo numa máquina cliente;
        - Permitem que os procedimentos tradicionais sejam executados em múltiplas máquinas a partir de chamadas pela rede de comunicação.
    - **RMI: Remote Method Invocation:**
        - RPC para o ambiente orientado a objetos;
        - Chamada de métodos em objetos remotos.
        - **RMI - Stub (Cliente):**
            - Transforma os parâmetros em formato independente de máquina (marshalling);
            - Envia requisições para o objeto remoto através da rede, passando o nome do método e os argumentos transformados.
        - **RMI - Esqueleto (Servidor):**
            - Recebe a requisição com o nome do método , decodifica (unmarshals) os parâmetros e utiliza-os para chamar o método no objeto remoto;
            - Transforma os dados resultantes e devolve-os para o cliente.

PPT 8:

- **Protocolo HTTP:**
  - **Funciona como protocolo de pedido-resposta:**
    1. Um browser envia um pedido HTTP para um servidor;
    2. O servidor retorna uma resposta HTTP para o cliente;
    3. A resposta contém dados de informação e o conteúdo requerido pelo cliente.
  - É um protocolo muito simples para tocar mensagens de texto entre duas máquinas.
- **HTTP 0.9:**
  - **Principais Objetivos:**
    - Permitir transferências de ficheiros de texto;
    - Permitir pesquisas em ficheiros hypertexto;
    - Permitir negociação de formatos de ficheiros;
    - Permitir referenciar outros servidores ao cliente.
  - **O protótipo inicial tinha algumas funcionalidades:**
    - O pedido do cliente é um conjunto de caracteres terminados com um CRLF;
    - A resposta é em HTML;
    - A ligação é terminada após a resposta.
- **HHTP 1.0:**
  - Surgiu no boom da internet para colmatar limitações anteriores:
    - Servir mais formatos de ficheiros, e não só HTML;
    - Providenciar metadados sobre os pedidos e respostas;
    - Permitir negociação de conteúdos;
    - ETC.
  - Nunca chegou a ser um standard.
  - **Métodos:**
    - **GET** -> Pede um recurso por URL;
    - **HEAD** -> Semelhante ao GET mas sem o conteúdo, apenas com os headers da resposta;
    - **POST** -> Requer ao servidor que aceite um certo conteúdo anexado.
- **HTTP 1.1:**
  - Santard que resolve ambiguidades das versões anteriores e otimiza a performance:
    - Reutilização de ligações;
    - Pedidos de bytes específicos;
    - Entre outros…
  - **Novos Métodos:**
    - **PUT** -> Requer ao servidor que aceite um certo conteúdo anexado, por URL, e se já existir é modificado;
    - **DELETE** -> Remove o recurso especificado no URL;
    - **TRACE** -> Faz eco o pedido recebido;
    - **OPTIONS** -> Retorna os métodos HTTP disponíveis;
    - **PATCH** -> Aplica alterações parciais ao recurso no URL;
    - **CONNECT** -> Encaminha o pedido.
  - **Limitações**:
    - Pedidos e respostas são sequenciais;
    - Para haver paralelismo tem de haver múltiplas ligações ao mesmo servidor;
    - Entre outras…
- **HTTP 2.0:**
  - Surgiu a partir do protocolo experimental SPDY da Google com os seguintes objetivos:
    - Melhorar a latência percebida sobre HTTP/1.1;
    - Não requerer múltiplas ligações paralelas;
    - Reter semântica do HTTP/1.1;
    - Entre outros…
  - É um protocolo binário.
  - **Ligações**:
    - Todas as comunicações são feitas numa ligação TCP;
    - Uma stream é um canal virtual dentro de uma ligação e tem um identificador;
    - Uma mensagem é uma mensagem HTTP, e pode estar dividida em vários frames.
  - **Multiplexagem**:
    - Permite enviar mensagens em várias streams paralelamente.
  - O protocolo HTTP/2.0 ainda está a ser adotado mundialmente.
- **HTTP/3:**
  - Apesar do protocolo HTTP/2 ter uma utilização mundial de cerca de 34%, já está a ser criada a versão 3;
  - Será baseado no protocolo experimental da Google (QUIC);
  - Utilizará o UDP como protocolo de transporte, invés do TCP.

PPT 9:

- **Nomes:**
  - **São Utilizados Para:**
    - Identificar entidades;

        - Referir localizações;
        - Partilhar recursos.
    - Sequências de caracteres para se referir a recursos.
    - Um recurso pode ter um ou muitos nomes, mas um nome apenas está associado a um recurso.
- **Endereços:**
    - Um recurso pode ter um ou mais pontos de acessos designados por endereços.
    - **Podem ser:**
        - Endereços IP; MAC ou de Memória.
- **Resolução de Nomes em Sistemas:**
    - **Nomeação Plana:**
        - Sistemas onde identificadores são nomes não-estruturados.
        - Soluções Simples:
            - **Brroadcast:**
                - Pedidos de resolução são enviados para todos os nós da rede;
                - Cada nó verifica se é o destinatário;
                - O destinatário responde com o seu endereço.
                - (Consome muitos recursos na rede)
            - **Multicast:**
                - Pedidos são enviados apenas para nós pertencentes ao grupo;
                - O destinatário responde com o seu endereço.
        - **Tabelas de Dispersão Distribuídas:**
            - Estrutura de dados que associa chaves de pesquisa a valores, onde cada nó tem apenas uma parte pequena da tabela total.
            - **Algoritmo baseado em tabelas de dispersão distribuídas:**
                - Cada nó tem um nó sucessor e um predecessor;
                - Cada nó reconhece n sucessores e n predecessores.
            - **Para resolver um nome:**
                - O nó inicial verifica se sabe o endereço do destinatário;
                - Senão, pergunta aos seus sucessores.
    - **Nomeação Estruturada:**
        - Nomes não estruturados tendem a ser pouco convenientes para redes mais complexas.
        - Nomes estruturados tendem a ser compostos por nomes simples e legíveis, organizados num namespace.
        - **Namespace:**
            - Pode agrupar os vários componentes do nome num formato tipo árvore:
                - **Raiz** -> Nome de mais alto nível;
                - **Diretórios** -> Subdividem um nome em subnomes;
                - **Folhas** -> Representam os recursos que se procura.
        - **Distribuição de Nomes:**
            - O spacename para redes de larga escala está normalmente associado hierarquicamente em **camadas lógicas**:
                - **Camada Global** -> Nós de mais alto nível e subnomes importantes;
                - **Camada Administrativa** -> Nós intermédios;
                - **Camada Final** -> Nós que representam os recursos.
        - **Resolução de Nomes Estruturados:**
            - Dado o nome de um recurso num namespace estruturado, o endereço é obtido por uma de duas formas:
                - **Iterativamente:**
                    - O servidor resolve a parte do nome que conseguir e devolve o restante ao cliente, que reencaminha o pedido para o próximo servidor.
                - **Recursivamente:**
                    - O servidor resolve a parte do nome que conseguir e reenvia o pedido a outro servidor, até completar a resolução do nome.
    - **Nomeação Baseada em Atributos:**
        - **Em certos casos, pode-se querer indexar os nomes dos recursos por atributos:**
            - Serviços que oferecem;
            - Capacidades que possuam;
            - Características que possuam.
        - **Diretórios de Serviços:**
            - Diretórios em que as entidades descrevem os seus atributos.

PPT 10:

- **Webservices**: Serviço criado de modo a suportar a interação entre máquinas sobre uma rede.
- **Funcionamento**: A aplicação solicita uma das operações disponíveis no webservice, e o webservice efetua o processamento e envia os dados para a aplicação.
- **Tecnologias Mais Usadas:**
    - **AJAX:**
        - Permite aos browsers obter informação de um servidor de forma assíncrona sem reler a página toda novamente.
    - **SOAP:**
        - Protocolo de mensagens que usa XML para trocar informação com um servidor.

    - **REST - Representational State Transfer:**
        - Arquitetura para interoperabilidade entre PCs na internet.
        - Servidores expõem serviços através de recursos na web.
        - É suposto ser mais leve que o SOAP e outros.
- **Em Suma:**
    - Webservices são serviços disponíveis na WEB;
    - São para ser consumidos por aplicações e não por utilizadores finais;
    - AJAX e REST são os tipos mais usados;
    - REST usa métodos HTTP para a sua semântica de interoperabilidade.