***TRABALHO DE PROCESSAMENTO PARALELO E DISTRIBUIDO***

***Prof. Dr. Sérgio T Castro***

***Aluno: Carlos Henrique Teles***

***PRIMEIRA PARTE DO TRABALHO***

**QUESTAO 1: Explique as diferenças entre o algoritmo Filtering-Based (Baseados em Filtro) e o algoritmo Rendez-vous, destacando os prós e contras de cada um, seja do ponto de vista dos subscribers, seja do ponto de vista dos Publisher.**

Os algoritmos acima descritos estão abaixo listados

1.1 Algoritmo para roteamento baseado em Filtragem de mensagens

upon receive publish(event e) from node x 1

matchlist := match(e, subscriptions) 2

send notify(e) to matchlist; 3

fwdlist := match(e, routing); 4

send publish(e) to fwdlist - x; 5

upon receive subscribe(subscription s) from node x 6

if x is client then 7

   add x to subscriptions; 8

else add(x, s) to routing; 9

  send subscribe(s) to neighbours - x;

1.2 Algoritmo baseado em rendez-vous

upon receive publish(event e) from node x at node i

rvlist := EN(e);

if i in rvlist then begin

  matchlist <- match(e, subscriptions);

  send notify(e) to matchlist;

  end

  send publish(e) to rvlist - i;

  upon receive subscribe(subscription s) from node x at node i

  rvlist := SN(s);

  if i in rvlist then

  add s to subscriptions;

else

  send subscribe(s) to rvlist - i;

Uma das estratégias mais simples para esses serviços de publicação é a baseada na inundação, ou seja, enviar uma notificação de evento para todos os nós da rede e, então, realizar a correspondência apropriada na extremidade assinante. O problema dessa estratégia, apesar da grande simplicidade de implementá-lo, e o grande volume de trafego desnecessário que o mesmo pode gerar na rede.

**QUESTÃO 2 – Se um modelo de comunicação é assíncrono, ele é também desacoplado no tempo (time-uncoupled)? Explique com exemplos.**

Quando nos referimos a comunicação síncrona ou assíncrona estamo-nos referindo a duas formas de intercâmbio da informação em função da simultaneidade com a que se envia e se oferece a mensagem.

No caso do sistema de comunicação ser assíncrono os participantes utilizam o sistema de comunicação em tempos diferentes. Correio eletrônico exemplifica bem esse caso, uma vez que o remetente de uma mensagem envia e o destinatário ler a mensagem em seu tempo, podendo ser quase que no mesmo momento que a mensagem foi enviada ou em um grande tempo depois.

**QUESTÃO 3 – A indireção é um conceito fundamental em ciência da computação, e, em termos de sistemas distribuídos, é muito aplicado nos paradigmas de comunicação. Explique as possibilidades de implementação de indireção em sistemas distribuídos e as duas propriedades decorrentes do seu uso.**

Nos sistemas distribuídos as tarefas são divididas entre as máquinas sem interação direta do usuário. Essa é principal diferença entre uma rede de computadores e um sistema distribuído, já que nas redes de computador todas as decisões de processamento e armazenamento é definido pelo usuário. Encontramos como características dos Sistemas Distribuídos a concorrência, a possibilidade de execução concorrente de tarefas, e caso seja necessário aumentar a capacidade de processamento ou armazenamento, basta adicionar novas máquinas ao sistema.

A comunicação indireta é definida como a comunicação entre entidades de um sistema distribuído por meio de um intermediário, sem nenhum acoplamento direto entre

o remetente e o destinatário (ou destinatários). Note que quando foi falado em destinatário, focou- se muito em que são vários, isso significa que muitos paradigmas da comunicação indireta suportam explicitamente a comunicação de um para muitos.

Duas propriedades que vem decorrentes do uso de comunicação indireta são:

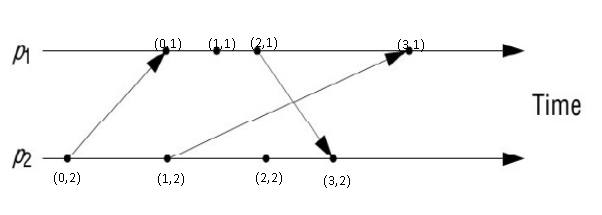
* Desacoplamento espacial, no qual o remetente não sabe ou não precisa saber a

identidade do destinatário (ou destinatários) e vice-versa.

* Desacoplamento temporal, no qual o remetente e o destinatário (ou destinatários)

podem ter tempos de vida independentes. Em outras palavras, o remetente e o destinatário (ou destinatários) não precisam existir ao mesmo tempo para se comunicar. Um exemplo desse paradigma de programação são as filas de mensagem (ou, mais precisamente, filas de mensagem distribuídas) que representam uma categoria importante de sistemas de comunicação indireta. Enquanto os grupos e os sistemas publicar-assinar fornecem um estilo de comunicação de um para muitos, as filas de mensagem fornecem um serviço ponto a ponto, usando o conceito de fila de mensagens como uma indireção, obtendo, assim, as propriedades desejadas do desacoplamento temporal e espacial.

**QUESTÃO 4 – A figura a seguir mostra eventos que ocorrem em dois processos p1 e p2. As setas entre os processos denotam a transmissão de mensagem. Identifique a malha de estados consistentes (p1, p2), começando com o estado inicial (0,0).**

  
  
Uma malha consistente nesses processos se dá iniciando pelo processo P2, em (0,2), depois para o processo P1 em (0,1), vai para (1,1) em P1 e finalizando em P2 em (3,2).

**QUESTÃO 5 – Explique, em termos dos relógios lógicos de Lamport, como ocorre uma ordenação total em um conjunto de eventos gerados por diferentes processos.**

Em muitas aplicações não interessa conhecer exatamente o tempo real, mas apenas que os vários processos concordem num determinado tempo **“ t ”**. Apesar de ser possível sincronizar os relógios de um sistema distribuído, essa sincronização não tem de ser absoluta.

Se dois processos não interagem, não é necessário que os seus relógios estejam sincronizados. O que muitas vezes é importante, não é qual o valor do tempo, mas que os processos concordem sobre a ordem pela qual os eventos ocorrem.

Para sincronizar relógios, Lamport introduziu a noção de precedência dos processos. Com a introdução da ideia de relógio lógico descreveu numericamente a noção de precedência sendo o relógio Lógico = contador em software que implementa uma função monótona crescente. Cada processo, **p**, terá o seu relógio lógico, **C**, que será usado para atribuir o tempo (timestamp) em que ocorre um evento.

O que se faz é implementar uma forma de medir o tempo tal que a cada evento **a** possa ser atribuído um tempo, **C(a)**, com que todos os processos concordem. Os valores do tempo têm de ser tais que: Se **a -> b** então **C(a) < C(b).**

Os processos executam em máquinas diferentes, cada uma com o seu relógio, cada relógio com a sua velocidade. O algoritmo de Lamport vai corrigir os relógios, de forma a verificar-se a ordem causal. Cada mensagem transporta consigo o tempo do processo emissor.

Algoritmo de Lamport:

- Seja **Li** o relógio lógico do processo **pi** , e **Li(e)** o timestamp do evento e no processo **pi** .

- Regra 1: **Li** é incrementado antes da atribuição de um timestamp a qualquer evento em **pi : Li := Li +1**

-Regra 2: a) Quando o processo **pi** envia a mensagem **m**, anexa (piggybacks) a **m** o valor **t = Li** (após o ter incrementado).

b) Ao receber uma mensagem **(m,t)**, um processo **pj** calcula **Lj := max (Lj , t )**, aplica a regra 1 e finalmente atribui um timestamp ao evento de recepção da mensagem

**QUESTÃO 6 – O funcionamento do algoritmo de eleição Baseado em Anel inclui a definição do estado participante e do estado não-participante para os processos. Qual é a importância de se estabelecer esse estado para cada processo? Explique.**

O Algoritmo de eleição baseado em Anel funciona da seguinte forma:

Processos são organizados em um anel lógico sendo que cada um possui um identificador. Então temos:

- Valores são únicos e completamente ordenados

- Identificadores podem ser dinâmicos. Por exemplo (identificador da máquina/ load);

- Cada processo sabe exatamente quem é seu sucessor;

- Todas as mensagens são passadas no sentido horário;

- O objetivo é escolher como coordenador aquele que possui maior identificador;

Cada processo pode iniciar a eleição colocando seu identificador em uma mensagem de eleição e mandando ao seu vizinho. Ele vai alterar então seu estado de não participante para participante.

Quando recebe uma mensagem de **eleição** compara seu id com o da mensagem

* + Se id da mensagem > que o seu id e seu estado é **participante**, simplesmente passa mensagem adiante.
  + Se id da mensagem > que o seu id e seu estado é **não participante**. Muda seu estado para participante e passa mensagem intocada
  + Se seu id > que o id da mensagem e estado **não participante**, muda seu estado para **participante** e repassa uma nova mensagem de eleição com seu id inserido

Quando recebe uma mensagem de **eleição** contendo seu próprio identificador venceu a eleição. Muda seu estado para **não participante** e envia uma mensagem com **eleito** contendo seu identificador.

Quando recebe uma mensagem **eleito** cada processo muda seu estado para **não participante** e muda suas variáveis internas para refletir o novo coordenador.

Então conclui-se que é de suma importância a mensagem que cada processo tem de participante ou não-participante, pois influencia no funcionamento do algoritmo para eleger qual processo vai coordenar.

**QUESTÃO 7 – Em que, essencialmente, o algoritmo de exclusão mútua de Maekawa difere do algoritmo Usando Multicast e Relógios Lógicos? Explique.**

No caso de multicast e relógios lógicos temos: Desenvolvido por Ricart e Agrawala para implementar exclusão mútua entre N processos pares usando multicast. Processos solicitam a entrada em uma Seção critica (SC) enviando uma mensagem para o grupo multicast e só podem entrar nela quando todos os outros processos do grupo responderem. As mensagens que solicitam entrada na SC são da forma: (T, Pi).

Onde: T é a indicação de tempo do remetente;

Pi é o identificador do remetente.

Cada processo registra seu estado em uma variável

- **RELEASED** – processo fora da SC;

- **WANTED** – deseja entrar na SC;

- **HELD** – encontra-se na SC.

Se um processo pede a entrada na SC e os estados dos outros processos for RELEASED, então todos responderão imediatamente ao pedido e o solicitante obterá a entrada. Se algum processo estiver no estado HELD, então esse processo não responderá aos pedidos até que tenha terminado com a SC.

Se um ou mais processos solicitarem entrada ao mesmo tempo, o pedido do processo que apresentar a indicação de tempo mais baixa será o primeiro a coletar as respostas garantindo a próxima entrada. Se apresentarem indicações de tempo iguais, serão ordenados de acordo com os identificadores correspondentes dos processos.

Já no caso do Algoritmo de Maekawa, temos:

Um processo não precisa da autorização de todos os outros processos para entrar na SC, Precisa obter permissão de apenas um subconjunto votante. Os subconjuntos usados por quaisquer dois processos precisam se sobrepor. Cada processo está em tantos conjuntos de votação quantos são os elementos em cada um desses conjuntos.

Então conclui-se que o que difere primordialmente dos anteriores é a quantidade de permissões necessárias para entrar em uma secção critica.

***SEGUNDA PARTE DO TRABALHO***

**“Sistema no qual componentes localizados em uma rede de computadores comunicam e coordenam suas ações somente através de passagem de mensagens”**

: 

**QUESTÃO 1: Explique a definição de Sistemas Distribuídos, discutindo o significado dos elementos das linhas 1 e 2 da tabela.**

Um sistema distribuído é um conjunto de computadores independentes entre si que se apresenta a seus usuários como um sistema único e coerente.

Vantagens de um Sistemas Distribuídos

● Melhor relação custo/benefício;

● Maior capacidade de processamento;

● Maior confiabilidade e disponibilidade;

● Crescimento gradativo de capacidade;

● Compartilhamento de recursos:– Hardware, software e dados.

Desvantagens de Sistemas Distribuídos

● Desenvolvimento de software adequado;

● Falhas e saturação da rede de comunicação:

– Podem eliminar as vantagens destes sistemas;

● Segurança pode ser comprometida:

– Acesso a dados e recursos reservados.

Características Básicas de Sistemas Distribuídos

**Heterogeneidade**

● Variedade e diferença em termos de:

– Hardware;

– Sistemas operacional;

– Rede;

– Linguagem de programação;

– Implementações de diferentes desenvolvedores.

● Exemplos de heterogeneidade na Internet:

– Diferentes implementações do mesmo conjunto de protocolos para diferentes tipos de rede: IP, TCP, UDP, SMTP

;– Diferentes padrões de representação de dados: IDL, XML;

– Diferentes padrões de bibliotecas: POSIX, DLL;– Diferentes padrões de invocação de serviços: COM, CORBA,RMI, SOAP;– Diferentes plataformas de execução: JVM (Java), CLR (.NET).

**Escalabilidade**

● Capacidade do sistema permanecer operando de forma efetiva mesmo diante de um aumento significativo do número de usuários e/ou dos recursos disponíveis;

● Principais desafios:– Controlar o custo dos recursos físicos;

– Controlar perdas de desempenho;

– Evitar “gargalos” de desempenho na rede ou nos próprios servidores.

**Transparência**

● Abstração/Ocultação para os usuários e programadores de aplicação:

– Sistema percebido como um “todo” coerente ao invés de uma coleção de partes independentes;

● Uma medida de sucesso de um sistema distribuído é dada pela sua transparência:

– Em que medida é indistinguível de um sistema centralizado com a mesma funcionalidade?

**Concorrência entre componentes**

Este termo não se refere a disputa, mais sim a execução simultânea de processos, utilizando computação paralela de processamento entre nós diferentes, porem com um mesmo objetivo.

**Ausência de um relógio global**

Não a sincronia entre clocks no envio e recebimento de mensagens. E interna a cada componente, o controle é feito por outros meios, por exemplo times tamps.

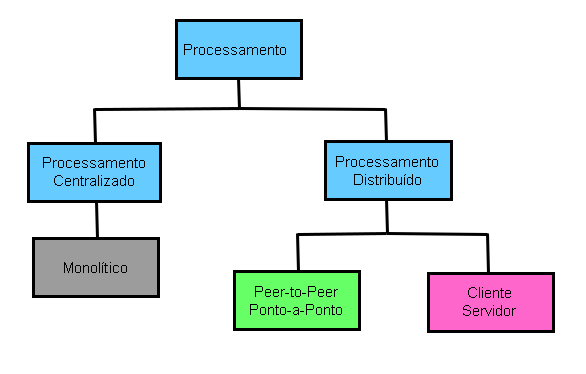
**Falha independente de componentes**

A queda não acarreta na falha de outros componentes. De forma que a solução seja o isolamento do problema de maneira invisível ao usuário.

**Arquitetura de Sistemas Distribuídos , Arquitetura de Servidores Concorrentes, Cliente-Servidor, Peer-to-Peer**

Um processo de software é um programa de computador sendo executado em uma CPU. “Processo”, “Tarefa” e “Serviço” podem ser considerados sinônimos. Processos concorrentes enviam e recebem mensagens entre si, prestando serviços uns aos outros para atender o usuário.

Um “sistema” Cliente/Servidor é composto por diversos destes processos, tarefas ou serviços.



**Processamento peer-to-peer**

* Não é hierárquico.
* Todos os processos são “clones” rodando em todas as máquinas e prestando serviços uns aos outros.
* Não existem processos servidores.
* Cada processo pode ser cliente e também servidor para outros processos simultaneamente.

Exemplo: Windows Explorer

**Processamento Cliente/Servidor**

* Existem processos distintos: o processo cliente é diferente do processo servidor.
* Processos cliente são sempre clientes.
* Processos servidores são sempre servidores.
* Cada processo é designado para a plataforma mais conveniente.
* Exemplo:
* Navegador (no cliente)
* Aplicação Web (no servidor)

**QUESTÃO 2: Explique o funcionamento dos elementos das linhas 3 e 4 da tabela**

**Representação externa de dados**

● Informações armazenadas em programas são representadas como uma estrutura de dados

● Na transmissão de informações, as informações são representadas numa seqüência de bytes

● Necessidade de conversão: estrutura de dados seqüência de bytes

● Problema – Diferentes sistemas possuem diferentes estruturas de dados – Ex: Código ASC x UNICODE, big endian x little endian, formatos de ponto flutuante etc.

● Solução – Conversão para um formato de dados de comum acordo (eXternal Data Representation – XDR)

– Enviar no formato do emissor e incluir informação sobre o formato empregado

**Formato independente de linguagem, SO etc**

● Utilizada para comunicação dos dados de requisições e respostas entre clientes e servidores

● Formato serializado

● **Marshalling**: conversão entre a representação interna e externa

➔ CORBA Common Data Representation

➔ Serialização de objetos em Java

➔ XML (Extensible Markup Language) Marshalling (Empacotamento) ● Processo de converter uma coleção de dados e organizá-los em um formato próprio para transmissão

– **Unmarshalling** é o processo oposto

● Ideal não haver o envolvimento explícito da aplicação (transparência) – Responsabilidade do middleware

● Duas técnicas básicas – Conversão dos dados para um formato binário – Conversão dos dados para um formato texto (ASC II)

● Serialização Java e XML

**Request-reply, IPC, RPC**

Comunicação em Sistemas Distribuídos.

Quando múltiplos processos fazem um trabalho conjunto, eles devem interagir

(IPC) “Comunicação interprocesso”: forma de interação ou comunicação entre processos.

Sistemas distribuídos possuem mecanismo de comunicação entre processos em diferentes máquinas (remotas), pois não há compartilhamento de memória física.

A base para um sistema distribuído é um processamento cooperativo de requisições submetidas por um cliente a um servidor que as processa e retorna um resultado. É uma forma especial de processamento distribuído em que os recursos estão espalhados em mais de um computador. O protocolo Request (Requisição) Reply (Resposta) é simples e direto.

RPC (Remote Procedure Call - Chamada de Procedimento Remota), é uma tecnologia de[comunicação entre processos](http://pt.wikipedia.org/wiki/Comunica%C3%A7%C3%A3o_entre_processos) que permite a um[programa de computador](http://pt.wikipedia.org/wiki/Programa_de_computador) chamar um[procedimento](http://pt.wikipedia.org/wiki/Subrotina) em outro[espaço de endereçamento](http://pt.wikipedia.org/wiki/Espa%C3%A7o_de_endere%C3%A7amento) (geralmente em outro computador, conectado por uma[rede](http://pt.wikipedia.org/wiki/Rede_de_computadores)). O programador não se preocupa com detalhes de implementação dessa interação remota: do ponto de vista do código, a chamada se assemelha a chamadas de procedimentos locais.

O objetivo do RPC foi permitir aos programadores se concentrar nas tarefas exigidas de um aplicativo, e ao mesmo tempo, tornando transparente para o programador o mecanismo que permite que as partes do aplicativo se comuniquem através de uma rede.

**RMI Programação com interfaces Semântica da chamada Transparência Chamadas locais e remotas Stubs e skeletons Interface remota**

Escolhas para a Semântica de Invocação RMI:

- Semântica de Invocação Talvez: o método remoto pode ser executado uma vez ou não ser executado.

- Semântica de Invocação pelo menos uma vez: o invocador recebe um resultado quando o método remoto foi executado pelo menos uma vez, ou recebe uma exceção, informando-o que nenhum resultado foi obtido. Pode ser obtida pela retransmissão das mensagens de requisição.

- Semântica de Invocação no máximo uma vez: ou o invocador recebe um resultado quando o método remoto foi executado exatamente uma vez, ou em caso contrário, uma exceção. É a semântica usada em Java RMI e CORBA, mas em CORBA é permitido a semântica talvez, usadas para métodos que não retornam resultados, e em RPC Sun é permitida a chamada pelo menos uma vez.

Os stubs e os skeletons são componentes intermediários, entre um cliente e o objeto. O stub é um componente do lado do cliente, que contém uma referência para o objeto remoto. A partir desta referência, chamamos o método do objeto remoto, via ORB. O skeleton é um componente do lado do servidor, e o ORB o usa para gerar a chamada de método remota em uma chamada local no objeto. O skeleton traduz a chamada e os parâmetros para a implementação específica do objeto. No retorno, o skeleton novamente traduz os resultados e erros gerados e os envia de volta para o cliente via ORBs.

**Middleware, CORBA, Serviços de Nomes, Referências de Objetos IDL Bind**

Middleware é um software que fornece um modelo de programação, e que é situado entre camadas de software (no nosso caso, o Java RMI, entre as camadas de aplicação e transporte), que usa um protocolo baseado em mensagens entre objetos (locais e remotos), no sentido de fornecer abstrações de mais alto nível, como as invocações e eventos remotos, proporcionando transparência de localização nas invocações remotas.

CORBA, da sigla Common Object Request Broker Architecture, é a especificação de um OMA (object management architecture) que tem por objetivo a interoperabilidade entre diferentes sistemas computacionais e linguagens de programação através de ORB’s, que são estruturas que permitem que os programadores façam chamadas de um computador a outro através de uma rede. O CORBA é definido e padronizado pela OMG.

Serviço de nomes e o componente da computação distribuída que permite que usuários e programas possam localizar e descrever pessoas, lugares, aplicações, e serviços em ambientes distribuídos. Ou seja, o Serviço de Nomes é o componente que permite que aplicações e serviços localizem e façam referência a objetos em um ambiente de computação distribuída. Os Nomes são uma parte crítica da computação distribuída, pois ajudam na administração dos recursos disponíveis na rede.

Objetos nunca são manipulados diretamente, mas sempre através de uma variável que contém uma referência para o objeto. Internamente, uma referência conterá o endereço para a área de memória que contém o objeto, mas isso é irrelevante sob o ponto de vista do programador. Alguns [exemplos de manipulação](http://www.dca.fee.unicamp.br/cursos/PooJava/objetos/manipul.html) de objetos usando a classe String devem esclarecer esses aspectos.

Em função dessa característica na manipulação de objetos, a atenção deve ser redobrada quando a aplicação precisa [comparar objetos](http://www.dca.fee.unicamp.br/cursos/PooJava/objetos/compara.html); a cópia de objetos também é afetada por esses mesmos aspectos.

Deve-se observar que o escopo da referência ao objeto encerra-se com o fim do bloco no qual a referência é definida, mas a remoção do objeto só ocorre através do processo de [coleta de lixo](http://www.dca.fee.unicamp.br/cursos/PooJava/objetos/remocao.html).

Uma referência especial é criada para que cada objeto possa poder referenciar a si próprio no corpo de seus métodos; esta referência é apontada pela variável implícita [this](http://www.dca.fee.unicamp.br/cursos/PooJava/objetos/this.html).