**Cliente/Servidor:**

Pode por ser uma arq. centr. ou descentr. de SD onde há processos que disponibilizam serviços (servidor) e outros que utilizam serviços (clientes). Tanto os clientes como os servidores podem estar em máquinas distintas e a computação deste paradigma é uma sequência de pares pedido/resposta. A conexão entre os dois é feita através de canais TCP onde os extremos da conexão são sockets onde o servidor fica à espera numa determinada porta e o cliente liga-se ao servidor através dessa mesma porta criando uma conexão bidirecional.

**Explicar a operação wait(cond, lock):**

As operações de wait(cond, lock) funcionam ao nível do SO com ajuda do hardware. Esta solução faz parte de um regime de exclusão mútua de modo a escalonar o acesso de threads às secções críticas. Quando uma thread é a 1º a chegar a uma SC que está rodeada por um lock X, adquire esse lock, todas as outras threads que invocaram a instrução wait com o lock X vão ter de esperar até que a thread que tem o lock X o libertar (unlock) e notificar todas as outras threads (notifyall). É necessário também alterar o valor da condição do wait.

**Funcionalidades de um servidor de objetos:**

Um servidor de objetos ao contrário de um servidor tradicional não oferece um serviço específico, mas mantém um conjunto de objetos que disponibilizam vários serviços. Tem como papel principal num SD de objetos, gerir um conjunto de objetos e intermediar os pedidos que são realizados aos objetos. É um servidor multi-thread podendo atribuir uma thread a cada objeto ou uma thread a cada invocação. Vantagem: Promove a exclusão mútua.

**Monitor é uma primitiva de alto nível:**

O monitor é uma primitiva estruturada de controlo de maior nível de abstração. O monitor é um tipo abstrato de dados com controlo de concorrência implícito em todas as operações com exclusão mútua, associa automaticamente os dados e operações ao código de controlo de concorrência e disponibiliza variáveis de condição implícita e/ou explicitamente. Desvantagem: O acesso aos métodos está sujeito a starvation.

**Algoritmo Descentralizado:** Quando um processo Pi pretende aceder à sua zona crítica gera uma etiqueta temporal TS e envia um pedido (Pi, Tsi) a todos os processos. Quando um processo recebe um pedido pode responder logo ou adiar a sua resposta. Quando um processo recebe respostas de todos os processos no sistema pode então executar a sua zona crítica. Depois de terminar a execução da sua zona crítica, o processo responde a todos os pedidos aos quais adiou a resposta. A decisão de Pj responder logo a um pedido (Pi, TSi) ou adiar depende de três fatores: Se Pj estiver na sua zona crítica, adia. Se Pj não pretender aceder à sua zona crítica, responde. Se Pj pretender aceder à sua zona crítica (e já enviou também um pedido): Se TSj > TSi então responde logo (Pi pediu primeiro) Senão adia a resposta.

**Algoritmo Centralizado:** Um processo é escolhido para coordenar o acesso à zona crítica. Um processo que queira executar a sua zona crítica envia um pedido ao coordenador. O coordenador decide que processo pode entrar na zona crítica e envia a esse processo uma resposta. Quando recebe a resposta do coordenador, o processo inicia a execução da sua zona crítica. Quando termina a execução da sua zona crítica, o processo envia uma mensagem a libertar a zona.

**Algoritmo Bully:** O algoritmo é despoletado por um processo Pi que julga que o coordenador falhou. Pi envia uma mensagem de eleição a todos os processos com maior prioridade que Pi. Se num intervalo T Pi não receber qq resposta, então autoelege-se coordenador. Se receber alguma resposta então Pi espera durante T' por uma mensagem do novo coordenador. Se não receber nenhuma mensagem então reinicia o algoritmo. Se Pi não é o coordenador então, a qualquer momento pode receber uma mensagem.

**Algoritmo em anel:** Aplicável a sistemas organizados física ou logicamente em anel. Assume que os canais de comunicação são unidirecionais. Cada processo mantem uma lista de ativos que consiste nas prioridades de todos os processos ativos quando este algoritmo terminar. Quando um processo Pi suspeita a falha do coordenador, Pi cria uma lista de ativos vazia, envia uma mensagem de eleição m(i) ao seu vizinho e insere i na lista de ativos.

**Escala geográfica e escala numérica:** Um dos objetivos dos SO é a escalabilidade do sistema que pode ser dada por uma escala geográfica, ou seja, distância máxima entre os nós, uma escala numérica, isto é, número de utilizadores e/ou processos e ainda número de domínios administrativos que diz respeito à escala administrativa. Relativamente às técnicas temos 3: mascarar as latências de comunicação (evitar esperar por respostas, comunicação assíncrona), distribuição (dividir esforço de computação, colocar computação nos clientes) e replicação e caching (disponibilizar cópias de dados em vários nós).

**Transparência de acesso e RPC:** Transparência de acesso mascara diferenças na representação dos dados na forma como os recursos são acedidos. O RPC é um modelo natural de programação que utiliza invocação de procedimentos e estes podem ser concebidos de forma a executarem isolados, sem dependência dos outros procedimentos, contribui para a transparência de dados, visto que a invocação de procedimentos remotos possibilita a execução de procedimentos em computadores distintos. Logo como vão haver diferentes computadores a aceder aos dados existindo diferenças na representação dos dados embora o utilizador não se aperceba.

**Relógios:** Existe um modo que permite sincronizar relógios em SD, é o algoritmo de sincronização de Clock – Algoritmo de Berkeley. Sabendo que nenhuma máquina tem um recetor de Tempo Universal Coordenado, este algoritmo cria um servidor de Tempo ativo que requer periodicamente o tempo de relógio de cada máquina para com isso calcular a média dos tempos e dizer à máquina como ajustar o seu relógio, diminuindo assim a sua incerteza.

**Lock/unlock ao nível do SO:**

As operações lock e unlock funcionam ao nível do sistema operativo com a ajuda do hardware. Estas soluções fazem parte do regime de exclusão mutua de forma a circundar secções críticas para escalonar o acesso das threads de um programa a essas mesmas regiões. Este tipo de primitivas de controlo de concorrência elimina esperas ativas. Quando uma thread adquire um lock, o kernel manda as outras threads “adormecerem”, isto é, coloca o estado delas a bloqueado até a thread que está a executar a secção crítica libertar o lock no fim, através de um unlock. Assim que o processo descrito libertar o lock (invocando o unlock), todos os outros processos que estavam adormecidos, passam do estado bloqueado para o estado pronto, sendo que ficam disponíveis para entrar na zona crítica se assim forem escalonados.

**Descrever o algoritmo distribuído de exclusão mutua em anel:**

O algoritmo distribuído de exclusão mutua em anel é aplicável a sistemas organizados física ou logicamente em anel. Neste algoritmo é assumido que os canais de comunicação são unidirecionais, cada processo mantém uma lista de ativos que consiste nas prioridades de todos os processos ativos quando este algoritmo terminar. Quando um processo Pi suspeita a falha do coordenador, Pi cria uma lista de ativos vazios, envia uma mensagem de eleição m(i) ao seu vizinho e insere i na lista de ativos. Por fim, se Pi recebe uma mensagem de eleição m(j) responde de uma de três formas: 1) se foi a primeira mensagem que viu, então cria uma lista de ativos com i e j e envia as mensagens de eleição m(i) e m(j), nesta ordem ao vizinho. 2) se i=!j então junta j à sua lista de ativos e reenvia a mensagem ao vizinho. 3) se i=j então a sua lista já contém todos os processos ativos no sistema e Pi pode determinar o coordenador.

**Diferenças ao assegurar exclusão mutua entre processos a executar numa única máquina ou num SD:**

A exclusão mútua entre processos numa única máquina está relacionado com não poder haver + que 1 processo a executar a secção crítica, pois como pode existir partilha de memória e recurso a dados partilhados, pode levar à inconsistência. Relativamente a exclusão mútua num SD cada processo executa no seu processador e caso esteja a aceder à secção crítica, então + nenhum processo o pode executar. Na 1ª abordagem para que seja assegurada são usadas primitivas de controlo de concorrência (lock e unlock) enquanto que para um SD são usados algoritmos centralizados (em que 1 processo coordena o acesso à secção crítica) e descentralizados (onde são utilizadas etiquetas temporais).

**Arquiteturas Descentralizadas de SD:** As arquiteturas descentralizadas apresentam-se como uma alternativa as arquiteturas centralizadas de sistemas distribuídos. A este tipo de arquitetura damos o nome de sistemas par-a-par ou peer-to-peer (P2P). Estes sistemas podem ser estruturas, nos casos em que obedecem a uma estrutura específica para a distribuição dos dados, ou podem ser não estruturados em que a interação entre pares é feita de forma aleatória. No primeiro caso, esta estrutura é obtida através do DHT (tabela de hash distribuída). No segundo caso, os sistemas não-estruturados funcionam de maneira mais aleatória, onde cada nó ligado à rede mantém uma vista parcial da mesma, consistindo num número razoavelmente pequeno de nós. Cada nó P escolhe periodicamente um nó Q da sua vista parcial. P e Q trocam informações e trocam nós das suas vistas parciais. A aleatoriedade com que as vistas parciais são mantidas é crucial para o funcionamento e robustez do sistema.

**Das Arquiteturas estudadas qual se adequa melhor a um sistema de suporte de redes sociais:** Em SD existem 4 tipos de arqs.: arq. em camadas, baseadas em dados, baseadas em objetos e em eventos. Para este tipo de implementações, o principal objetivo não é guardar dados (no entanto também o faz), mas sim permitir aos utilizadores verem tudo aquilo que os outros fazem, ou seja, baseada numa arq. de eventos. Neste tipo de arqs. os processos comunicam através de eventos, os quais opcionalmente pode carregar dados.

**Diga o que entende por sincronização de processo em programação concorrente:** Em programação concorrente, os processos a executar são ações autónomas, a velocidade de cada ação é imprevisível, pois não existe maneira de saber quais as velocidades relativas. Dentro da programação concorrente existem 2 paradigmas: a comunicação e a sincronização. Neste caso, a sincronização pode estar ou não associado a comunicação, uma vez que os processos podem ter de esperar antes de prosseguir. Isto pode levar a esperas ativas, coisa que queremos evitar.

**Midleware orientado as mensagens:** O Midleware é uma camada de software que se estende por várias máquinas fornecendo uma abstração para a programação de aplicações e o seu papel é o de melhorar a transparência que um sistema distribuído deve ter, facilitar o acesso a recursos remotos/distribuídos, ter abertura e extensibilidade e escalabilidade. O midleware orientado às mensagens utiliza comunicação assíncrona, isto é, o emissor não sincroniza com o recetor e é persistente, ou seja, as mensagens são guardadas até serem entregues, através de filas de mensagens. Os sistemas de filas de mensagens assumem um protocolo de dados e da sua estrutura. No entanto, uma das principais aplicações dos sistemas de filas de mensagens é integrar diferentes aplicações desenvolvidas independentemente num sistema de informação distribuída coerente.

**Processo:** Dois processos não partilham a memória (mesmo que seja pai e filho), ou seja, são criados da mesma maneira, mas se a variável do pai é alternada, a do filho não irá mudar. Cada processo tem o seu espaço de endereçamento privado.  
**Thread:** podem ser vistas como processos leves que permitem cooperação eficiente via memória; Podem também ser pedidos ao SO segmentos de memória partilhada entre processos;

**Baseada em Objetos:** É usada quando não é eficaz usar por camadas, mas sim por um conjunto de entidades, que têm funções atómicas e que as vamos modelar e estabelecer ligações entre elas; Isto é feito á custa da noção de objetos. Usamos um application Server. Existe encapsulamento do estado dos objetos. Difícil de desenvolver.

Baseada em Dados: É usada quando a aplicação depende de um grande número de dados. O centro do sistema é uma grande infraestrutura partilhada onde temos dados recolhidos. A funcionalidade é através de componentes funcionais que consomem dados.

**Sistemas de Ficheiros Distribuídos:** Os sistemas de ficheiros distribuídos assentam tipicamente em RPCs. Em redes de larga escala (distribuição geográfica) têm que ser tidas em conta algumas optimizações: agrupamento (batching) de comandos e caching de ficheiros ou blocos. **Semântica Partilha Ficheiros:** UNIX: uma leitura devolve o valor da última operação de escrita. Apenas se consegue obter em modelos de acesso remoto em que há apenas uma cópia dos ficheiros. TRANSACIONAL: o sistema oferece suporte para transações sobre ficheiros individuais. A implementação levanta grandes desafios para permitir o acesso a ficheiros fisicamente distribuídos. SESSÃO: os efeitos de operações de escrita e leitura são visíveis pelo cliente que abriu uma sessão de acesso ao ficheiro. A implementação de controlo de sessões concorrentes poderá ser conservadora (locks) ou otimista (certificação no fecho das sessões).