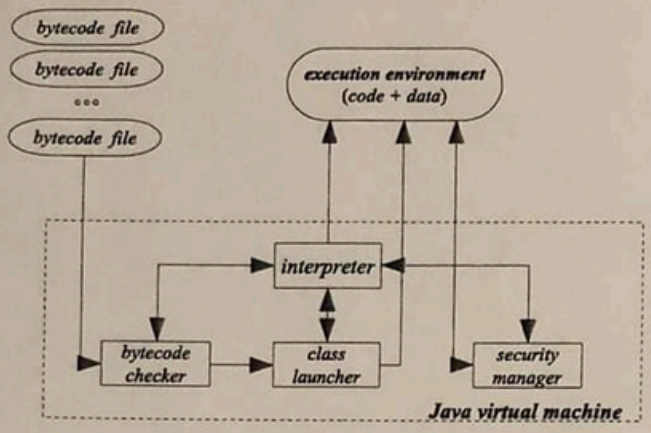
Part A (12 points)

1. The diagram below describes the main components of the Java Virtual Machine (JVM).



Define *middleware* and explain why the Java Virtual Machine (JVM) can be considered part of the middleware. Present clearly your claims.

(1) O middleware é uma camada de compatibilidade que permite que diferentes dispositivos num mesmo sistema possam correr independentemente do seu sistema operativo, hardware.

Ao ser implementado cria uma de abstração do hardware, sistema de rede e sistema operativo

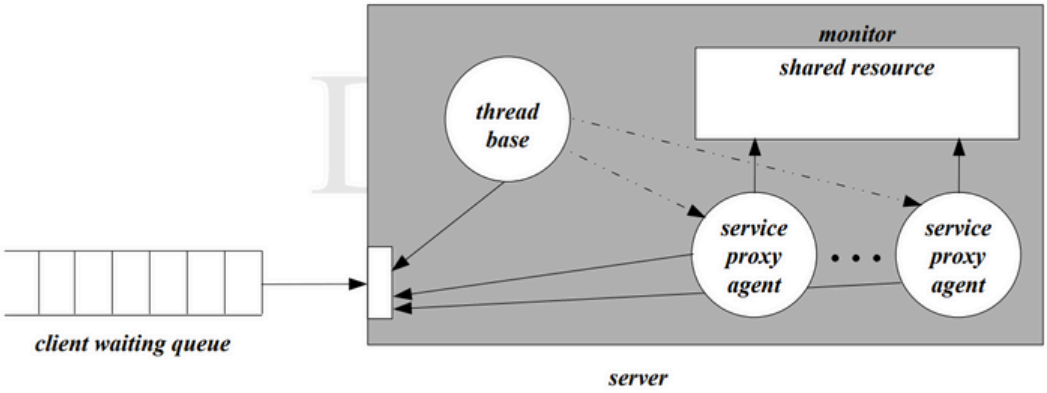
O JVM (Java Virtual Machine) é considerado um middleware pois implementa um sistema de segurança de 3 camadas

-> bytecode checker verifica a conformidade do bytecode com as regras de segurança e carrega as classes dinamicamente.

-> O security manager - Controla o acesso da aplicação aos recursos do sistema, como ficheiros, rede

Porisso, como o JVM gera bytecode que pode ser executado independentemente do SO, garante a compatibilidade e segurança.

2. Take *message passing* as the communication paradigm which was chosen to establish a client-server model among processes residing in different nodes of a distributed system. Draw a diagram that describes the functional interaction among the components of such a model, both at client and at the server side, identifying each of them and explaining their role in the interaction. Assume the server replication variant.
Refer in this context to what one means by *message marshaling* and *unmarshaling*. How are they enforced in Java?



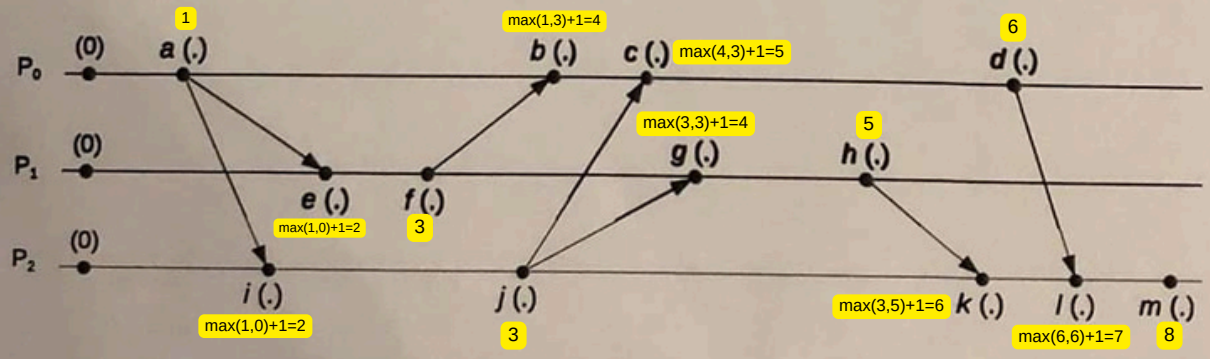
A variante de cliente-servidor denominada "Server Replication" consiste em replicar os servidores para aumentar a disponibilidade, a tolerância a falhas e o desempenho do sistema. Nesse modelo, várias réplicas de um servidor são mantidas, cada uma delas capaz de atender às solicitações dos clientes.

Num contexto de um modelo cliente-servidor baseado na variante Server Replication, onde é aplicado o paradigma de passagem de mensagens, uma mensagem é transmitida através de um canal de comunicação específico. Como observado no esquema, as mensagens aguardam numa fila (queue) para serem posteriormente transmitidas. Antes que isso ocorra, o receptor (servidor) deve saber como interpretar cada mensagem, recebida como bytes e posteriormente convertida em valores concretos.

Assim, a mensagem deve incluir não apenas os seus valores, mas também o tipo dos valores transmitidos, a sua estrutura, entre outros detalhes. A operação de construção de uma mensagem com todas essas informações incluídas é denominada "marshaling" de dados, e a operação contrária, que recupera todas essas informações para que a mensagem original seja reconstituída, é chamada de "unmarshaling" de dados.

No Java, o "marshaling" e o "unmarshaling" de dados são ocultados do programador, sendo apenas necessário que o tipo de dados da mensagem seja previamente definido como implementando a interface Serializable.

3. The schematics below depicts the *temporal* evolution of three processes whose local clocks are scalar logical clocks synchronized according to the Lamport algorithm.



i. Assign to the different events, specified by small letters (*a* ... *m*), their associated time stamp.

ii. Take from the schematics three examples of pairs of concurrent events (||) and list the longest sequence of sequential ones (->).

iii. Can the interaction depicted in the schematics be used as an example of access to a critical region with mutual exclusion in a distributed way? Present clearly your claims.

3.ii)

Concorrent Pairs:

Não existe conexão direta entre o par (b,g) , pelo que se trata de um par concorrente: b || g

Não existe conexão direta entre o par (c,h) , pelo que se trata de um par concorrente: c || h

Não existe conexão direta entre o par (f,j) , pelo que se trata de um par concorrente: f || j

Longest Sequence of Sequential Pairs:

a -> i

i -> j

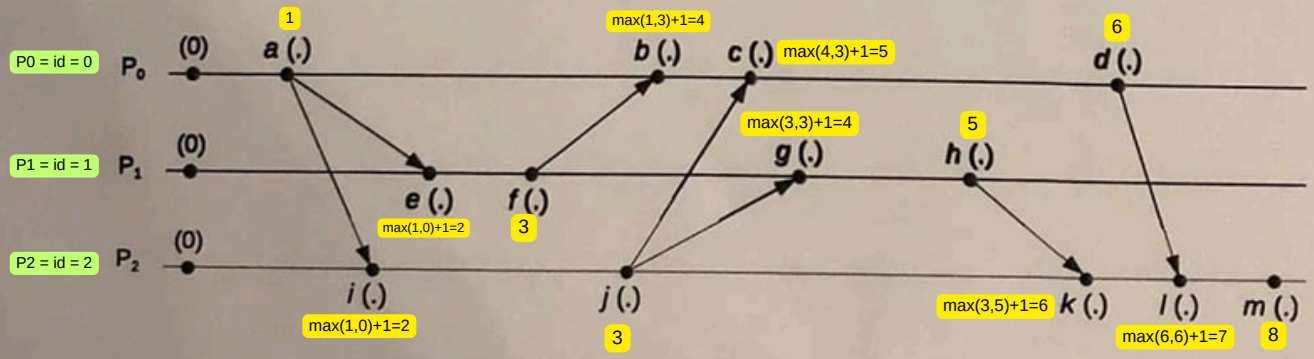
j -> c

c -> d

d -> l

l- > m

3. The schematics below depicts the *temporal* evolution of three processes whose local clocks are scalar logical clocks synchronized according to the Lamport algorithm.



i. Assign to the different events, specified by small letters (*a* ... *m*), their associated time stamp.

ii. Take from the schematics three examples of pairs of concurrent events (||) and list the longest sequence of sequential ones (->).

iii. Can the interaction depicted in the schematics be used as an example of access to a critical region with mutual exclusion in a distributed way? Present clearly your claims.

3.iii)

Para determinar se este é ou não um exemplo de acesso a um recurso partilhado com exclusão mútua de forma distribuída, é necessário inicialmente analisar se os valores escalares podem ser usados para estabelecer uma ordem total de eventos e se existe um mecanismo que garanta a exclusão mútua.

Para este efeito, pode ser implementado um relógio lógico escalar capaz de atribuir um ID único a cada evento ocorrido. Deste modo, é possível ordenar totalmente os acessos ao recurso partilhado em questão, fornecendo-lhe as timestamps resultantes do algoritmo de Lamport, bem como os IDs do processo a que cada evento está associado. Este relógio conseguirá assim processar esses valores e realizar o processo de ordenação total, desempatando timestamps iguais com base nos diferentes IDs de processo. Por exemplo, neste caso, o evento e pertencente ao processo 1 possui timestamp=2, tal como o evento i pertencente ao processo 2. Neste caso, como 1<2, a prioridade seria atribuída ao evento e.

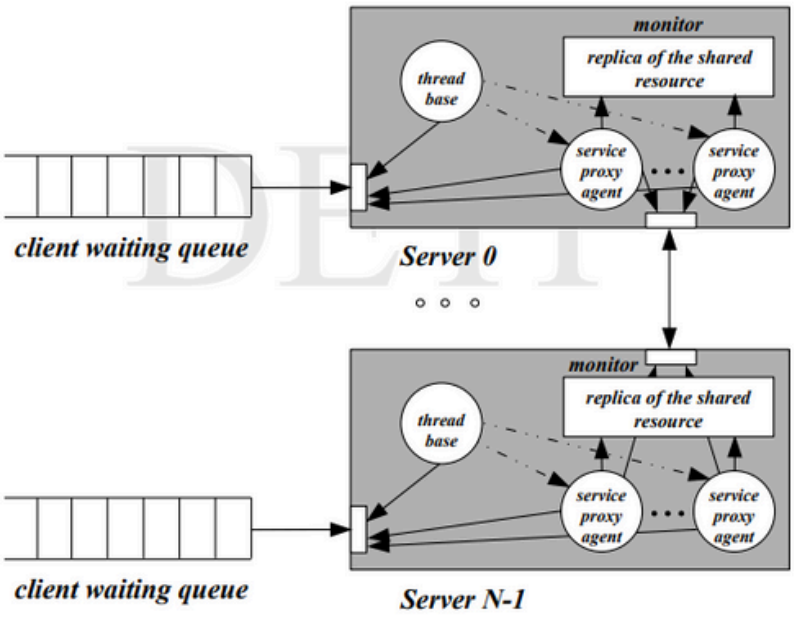
Para garantir a exclusão mútua, além da ordenação total dos eventos, é necessário implementar um protocolo específico de exclusão mútua distribuída, nomeadamente o Algoritmo de Ricart-Agrawala que se baseia na troca de mensagens entre processos de modo a garantir que só um processo por vez tenha acesso a um recurso partilhado. Cada processo enviará uma mensagem de pedido para todos os outros processos, incluindo o seu próprio timestamp, ao que os restantes processos respondem com uma mensagem de permissão a menos que os próprios estejam a utilizar o recurso ou tenham pedido acesso anteriormente com timestamp menor.

Sistemas Distribuídos
Teste 1

20 de Junho de 2018

4. Suppose we want to ensure *load balancing* of the server tasks in a *client-server* model. In order to achieve this aim, a *resource replication* variant where the server is installed in two hardware platforms, is implemented. Both server instantiations are active at the same time and interact with the clients. Draw a functional diagram that depicts the organization and list three problems which must be solved for the system to work properly. Justify clearly your claims.

Resource replication



O sistema deverá conseguir distribuir os pedidos de maneira a que nenhuma máquina especifica fique sobrecarregada ou subutilizada, trabalhando assim de acordo com uma politica ou algoritmo previamente definido.

O sistema deverá ainda conseguir gerir e sincronizar corretamente os diferentes recursos partilhados, mantendo assim alguma redundância para tolerância a falhas.

Deverá ainda ser capaz de lidar com falhas, ou seja, no caso de falha de uma das máquinas, depois da falha ser efetivamente detetada, a carga de trabalho deverá ser corretamente distribuida.