TSR: Segon Parcial

Aquest examen consta de 20 qüestions d'opció múltiple. En cadascuna, només una resposta és correcta. Ha de respondre's en una altra fulla. Les respostes correctes aporten 0.5 punts a la qualificació de l'examen. Les errònies descompten 0.167 punts.

TEORIA

1. El desplegament inclou la instal·lació inicial i la configuració d'una aplicació. A més d'aquestes tasques, el desplegament d'un servei també comprèn…

A	La depuració dels programes. Fals. L'etapa de depuració ha de ser aplicada acuradament abans que els programes es distribuïsquen i puguen ser instal·lats. El desplegament comprèn aquelles etapes posteriors a la distribució del programari, la primera de les quals seria la seua instal·lació. Per tant, la depuració no forma part del desplegament.
B	La gestió del cicle de vida del servei. Cert. La gestió del cicle de vida d'un servei inclou aquelles tasques aplicades a un servei després de la seua instal·lació (per exemple, la seua activació, desactivació, eliminació, etc.). Aquestes tasques estan incloses en el desplegament.
C	El desenvolupament de l'aplicació. Fals. Tal com s'ha suggerit en el primer apartat (posat que el desenvolupament dels programes precedeix la seua depuració), el desenvolupament no forma part del desplegament.
D	El disseny de l'aplicació. Fals. Com el disseny precedeix el desenvolupament i la depuració, tampoc forma part del desplegament.

2. L'objectiu principal de la injecció de dependències és…

A	Resoldre les dependències entre components utilitzant fitxers de configuració. Fals. La injecció de dependències consisteix a resoldre les dependències entre components de forma diferida i transparent, amb una intervenció mínima (idealment, cap) per part dels administradors. Els fitxers de configuració requereixen intervenció de l'administrador. Per tant, ha d'evitar-se el seu ús en la injecció de dependències.

B	Eliminar totes les dependències entre components durant l'etapa de disseny. Fals. Els components necessiten informació (interfícies, "endpoints"…) sobre els altres components per a poder interactuar amb ells. Algunes d'aquestes dades només podran ser conegudes durant el desplegament (per exemple, els "endpoints" d'altres components) o una vegada el component ja estiga funcionant (per exemple, el nou "endpoint" d'un component que s'haja migrat, el node servidor d'una base de dades responsable de certa clau primària quan s'utilitze particionat horitzontal, etc.).
C	Que la resolució de dependències siga el més transparent possible per al desenvolupador dels components. Cert. La injecció de dependències consisteix a facilitar la resolució de dependències durant l'execució dels components, enllaçant aquests dinàmicament amb objectes que mantinguen la informació necessària. Aquests objectes es comporten com "proxies": faciliten la interfície adequada i ofereixen una imatge local que oculta la interacció amb altres components externs, evitant així que el programa client haja de dedicar esforços a la resolució de dependències.
D	Evitar l'ús de contenidors perquè aquests penalitzen el rendiment. Fals. Els contenidors automatitzen alguns passos de resolució de dependències. Així, són una ajuda per a implantar diferents mecanismes d'injecció de dependències.

3. **Imaginem un servei que necessite 400 ms per a processar localment cada petició que modifique el seu estat. Aquestes peticions inverteixen 20 ms per a transmetre a altres rèpliques l'estat modificat i 30 ms a aplicar aquestes modificacions en elles. Un missatge de petició pot ser difós (en ordre total) en la xarxa en 3 ms. Una petició de lectura pot ser gestionada localment en 20 ms. La proporció d'accessos és: 80% accessos de lectura i 20% accessos de modificació.**
Per a escalar aquest servei, la millor aproximació serà…

A	Replicar-lo utilitzant el model actiu. Fals. En el model actiu totes les rèpliques dedicaran 400 ms per a processar localment cada petició de modificació. El model passiu solament introdueix aquesta càrrega en la rèplica primària. Les altres només hauran de dedicar 30 ms de processament local per a aplicar les modificacions. Per això, les rèpliques secundàries només necessiten dedicar 1/13 del temps utilitzat tant per les rèpliques del model actiu com per la rèplica primària per a gestionar les modificacions. Així, si un servei es desplega utilitzant quatre rèpliques, totes (en el model actiu) o la primària (en el model passiu) tindran una càrrega alta per a gestionar sol·licituds de modificació. Aquesta càrrega serà comparativament molt baixa en les rèpliques secundàries del model passiu. Aquestes rèpliques secundàries podran ser utilitzades com a primàries per a altres serveis replicats perquè un alt percentatge de la seua capacitat de còmput romandrà lliure. Per tant, el model actiu no és una bona aproximació per a gestionar aquest servei replicat.

B	Replicar-lo utilitzant el model passiu, processant totes les sol·licituds en la rèplica primària. Fals. Encara que aquesta és una alternativa millor que l'anterior, la que es descriu en l'apartat C és encara millor.
C	Replicar-lo utilitzant el model passiu, però permetent que les sol·licituds de lectura siguen processadas per les rèpliques secundàries. Cert. Amb aquesta configuració la càrrega pot equilibrar-se millor entre totes les rèpliques. Observe's que en el model actiu totes les peticions han de ser propagades a totes les rèpliques per a superar així les fallades arbitràries. Aleshores, tant en el model actiu (apartat A) com en el passiu estricte (apartat B) no s'arriba a donar cap distribució de càrrega entre les rèpliques.
D	No replicar-lo perquè la replicació introdueix massa coordinació i això impedeix un escalat eficient. Fals. Els serveis distribuïts han de superar les situacions de fallada. Si no es replicara aquest servei, s'estaria introduint un punt únic de fallada allí on es desplegue el servei. Encara que la replicació introduïsca cert nivell de coordinació, les peticions de consulta poden distribuir-se entre les rèpliques (vegeu la descripció de l'apartat C) i, així, es podrà escalar el servei de manera eficient. Sense replicació, una instància hauria d'atendre totes les peticions. Així, l'única instància podria saturar-se fàcilment.

4. En els models de consistència centrats en dades, podem dir que el model A és més fort que el model B en els següents casos:

A	A: causal, B: "*cache*". Fals. Els models causal i "*cache*" no poden comparar-se. Hi ha execucions que són causals però no "*cache*" i unes altres que són "*cache*" però no causals. Un model A és més fort que un altre B quan totes les execucions d'A també respecten B i hi ha execucions que respecten B però no A.
B	A: FIFO, B: "*cache*". Fals. Els models FIFO i "*cache*" no poden comparar-se. Hi ha execucions que són FIFO però no "*cache*" i unes altres que són "*cache*" però no FIFO.
C	A: causal, B: seqüencial. Fals. El model seqüencial és més fort que el causal.
D	A: causal, B: FIFO. Cert. El model causal és más fort que el FIFO; és a dir, totes les execucions causals són també FIFO, però hi ha execucions FIFO que no són causals. Per exemple, en un sistema amb tres processos, aquesta execució… W1(x)1, R2(x)1, W2(x)2, W1(x)3, R2(x)3, R3(x)2, R3(x)1, R3(x)3, R1(x)2, … …és FIFO perquè P1 ha escrit dos valors diferents (1 i 3, en aquest ordre) sobre "x" i P3 i P2 han rebut tots dos valors en l'ordre d'escriptura. Com no hi ha altres processos que hagen escrit més d'una vegada, això significa que totes les condicions de la consistència FIFO han sigut respectades en l'execució. No obstant això, P2 va escriure el valor 2 una vegada havia llegit el valor 1. Això implica que tots dos valors estan causalment relacionats (1 → 2), però P3 no els va obtenir en l'ordre causal. Per tant, l'execució no és causal i això demostra que el model FIFO és més relaxat que el causal.

5. Els magatzems escalables NoSQL no suporten el model relacional perquè…

A	El model relacional no admet replicació. Fals. Les bases de dades relacionals poden replicar-se, com qualsevol altre servei distribuït. Per exemple, hi ha múltiples edicions de MySQL, Oracle o Microsoft SQL Server (entre altres companyies de programari) que repliquen les seues bases de dades.
B	El model relacional no admet particionat horitzontal ("*sharding*"). Fals. Les bases de dades relacionals poden particionar-se horitzontalment. De fet, les primeres propostes de particionat horitzontal es van fer sobre bases de dades relacionals replicades.
C	Les dades relacionals han de mantenir-se en disc. Fals. Les dades han de mantenir-se en disc en aquest tipus de bases de dades, però això no impedeix que s'apliquen múltiples tècniques d'escalat.
D	Les transaccions en el model relacional necessiten mecanismes de control de concurrència que poden ser complexos. Cert. Les transaccions utilitzades en les bases de dades relacionals es compliquen quan s'introdueix replicació. Per exemple, per a finalitzar una transacció es necessita un algorisme distribuït (2PC o 3PC) i aquest algorisme introdueix una interacció forta entre els agents participants. Addicionalment, els mecanismes de control de concurrència poden necessitar la seua extensió a una gestió distribuïda (utilitzant "locks" distribuïts, per exemple), i això complica la seua gestió i introdueix interacció forta entre els processos participants. Per això, els magatzems NoSQL han renunciat a les transaccions amb garanties ACID per a millorar la seua escalabilitat.

6. El teorema CAP…

A	…exigeix que els serveis escalables i disponibles utilitzen sempre el model de consistència estricte per a tolerar així les particions de la xarxa. Fals. Al contrari, el teorema CAP diu que la consistència estricta no pot mantenir-se en serveis que pretenguen ser altament disponibles mentre es donen particions en la xarxa.

B	...permet que els serveis escalables relaxen la seua consistència mentre la xarxa romanga particionada, assegurant així la seua disponibilitat. Cert. El teorema CAP diu que els serveis distribuïts no poden suportar simultàniament consistència forta, disponibilitat i tolerància a les particions de la xarxa. Així, quan la xarxa es particiona ha de triar-se entre disponibilitat o consistència forta. Si un servei prefereix estar disponible ha de relaxar la seua consistència mentre la xarxa estiga particionada.
C	...no permet la implantació de serveis altament disponibles utilitzant models de consistència forts. Fals. Els models de consistència forta poden utilitzar-se en serveis distribuïts. El que diu el teorema CAP és que si un servei vol mantenir una consistència forta mentre la xarxa es particione, llavors haurà de renunciar a la seua disponibilitat en totes les rèpliques. Això significa que en subgrups minoritaris no podrà respondre's a les peticions dels clients i, per tant, aquestes rèpliques no estaran disponibles. De fet, el teorema CAP si que tolera alta disponibilitat i consistència forta. Per a fer això, el servei ha de desplegar-se de tal manera que es garantisca que no hi haja particions en la xarxa. Per exemple, en desplegaments per a uns pocs processos clients amb poques rèpliques servidores en un mateix laboratori i amb la xarxa de comunicacions replicada.
D	...no té sentit en els centres de dades de computació en el núvol perquè no hi haurà mai particions de la xarxa en ells. Fals. La computació en el núvol ofereix la imatge d'una escalabilitat il·limitada. Per a fer això, poden utilitzar-se múltiples centres de dades per a desplegar alguns serveis. Fins i tot en cas d'utilitzar un sol centre de dades, el centre tindrà molts "racks" d'ordinadors i molts segments de xarxa. En aquests casos es poden seguir donant particions en la xarxa. Per tant, les restriccions establides en el teorema CAP encara són aplicables en aquest tipus de desplegaments.

7. Respecte a l'escalabilitat de serveis es pot afirmar que…

A	Un mateix servei no pot escalar horitzontalment i vertical. Fals. Les escalabilitats horitzontal i vertical no són mútuament excloents.
B	Els algorismes descentralitzats milloren l'escalabilitat de distància. Fals. Els algorismes descentralitzats milloren l'escalabilitat de grandària, però no necessàriament l'escalabilitat de distància.
C	El particionat horitzontal ("*sharding*") millora l'escalabilitat administrativa. Fals. El particionat horitzontal millora l'escalabilitat de grandària, però no simplifica ni millora l'escalabilitat administrativa.
D	Evitar la contenció és un factor clau per a millorar l'escalabilitat d'un servei. Cert. La contenció evita que els serveis escalen ja que bloqueja l'execució d'aquests serveis.

8. Els objectius principals d'un subsistema de seguretat són:

A	Protecció, control d'accés i seguretat física. Fals. El control d'accés i la seguretat física no són objectius. Són mecanismes que poden utilitzar-se per a implantar polítiques de seguretat.

B	Protecció, gestió de la confiança i un bon suport per a mecanismes de xifrat. Fals. El xifrat és un mecanisme a utilitzar per a millorar la confidencialitat i integritat d'un sistema. La gestió de la confiança és un mecanisme per a assegurar i avaluar la correcció d'un sistema de seguretat. No formen part dels objectius sinó del conjunt de mecanismes.
C	Comptabilitat, integritat, confidencialitat i disponibilitat. Cert. Aquests són els quatre objectius que s'han presentat en el Tema 8 per als subsistemes de seguretat.
D	Polítiques robustes, mecanismes eficients i garanties correctes. Fals. Aquests són els tres tipus d'eines necessaris per a especificar, implantar i avaluar la correcció d'un sistema de seguretat, respectivament.

SEMINARIS

9. Quina de les següents tasques NO es realitza en utilitzar aquesta ordre Docker?

```
docker run –it ubuntu /bin/bash
```

A	Executar el programa /bin/bash en un contenidor. Fals. L'últim argument en aquesta línia d'ordres indica que el programa a executar en el contenidor serà "/bin/bash".
B	Descarregar la imatge "ubuntu:latest" des de Docker Hub si no la teníem en el dipòsit d'imatges local. Fals. L'argument "ubuntu" especifica el nom de la imatge a utilitzar per a generar el contenidor. Quan no s'especifica cap etiqueta, Docker assumeix "latest". Per tant, la imatge a cercar serà "ubuntu:lastest". Docker cerca aquesta imatge en el dipòsit local. Si no està allí, Docker la descarregarà del Docker Hub i la mantindrà en el dipòsit local.
C	Arreplegar l'eixida del contenidor que està sent utilitzat per a executar l'ordre. Aquesta eixida pot mostrar-se mitjançant l'ordre `docker logs`. Fals. L'eixida del contenidor és emmagatzemada pel servidor Docker i pot ser mostrada mitjançant l'ordre "docker logs" utilitzant l'identificador o nom del contenidor com a argument.
D	Eliminar automàticament aquest contenidor una vegada la seua execució haja acabat. Cert. Perquè faça això, la línia d'ordres hauria d'incloure l'opció "--rm=true". Sense ella, el contendor es mantindrà una vegada finalitze la seua execució.

10. L'ordre `docker commit a b` …

A	Crea un nou contenidor anomenat "a" utilitzant el Dockerfile situat en la carpeta "b". Fals. Per a generar un nou contenidor ha d'utilitzar-se l'ordre "docker run" o "docker create".

B	Crea una nova imatge “b” utilitzant el Dockerfile de la carpeta “a”. Fals. Per a crear una nova imatge a partir d'un Dockerfile s'ha d'utilitzar l'ordre “docker build”.
ℂ	Crea una nova imatge “b” amb el contingut actual del contenidor el nom o identificador del qual és “a”. Cert. Aquesta és una descripció breu del que fa l'ordre “docker commit”.
D	Realitza el “commit” d'una transacció “a” que va ser iniciada amb una ordre `docker pull` o `docker push`, generant un contenidor amb ID “b”. Fals. Docker no utilitza transaccions.

11. El mòdul “cluster” de NodeJS s'utilitza per a…

A	…desplegar un conjunt de programes NodeJS en un “*cluster*” d'ordinadors. Fals. Els mòduls NodeJS s'utilitzen durant l'etapa de desenvolupament. No s'utilitzen durant l'etapa de desplegament.
B	…gestionar múltiples fils en un procés NodeJS. Fals. Els programes JavaScript només tenen un únic fil d'execució. És impossible gestionar múltiples fils en aquest llenguatge i NodeJS és un intèrpret de JavaScript.
ℂ	…gestionar un conjunt de processos NodeJS perquè puguen compartir alguns recursos; p. ex., un port en el qual escoltar i un mateix programa a executar. Cert. Aquest és l'objectiu del mòdul.
D	…implantar fàcilment múltiples models de consistència de memòria. Fals. No està dissenyat per a aquesta fi.

12. MongoDB utilitza els següents mecanismes per a millorar la seua escalabilitat:

A	Control de concurrència distribuït. Fals. El control de concurrència distribuït és més complex i introdueix major contenció que el control de concurrència local. Per això no millora l'escalabilitat.
B	Algorismes descentralitzats. Fals. Encara que els algorismes descentralitzats són un bon mecanisme per a millorar l'escalabilitat, no s'ha descrit cap algorisme descentralitzat en la documentació sobre MongoDB.
ℂ	Replicació passiva i particionat horitzontal (o “*sharding*”). Cert. MongoDB utilitza replicació passiva i particionat horitzontal (incrementant així la seua capacitat per a atendre concurrentment múltiples peticions clients sobre diverses instàncies MongoDB) per a ser escalable.
D	Escalabilitat administrativa. Fals. No s'ha descrit cap mecanisme de configuració de MongoDB per a millorar la seua escalabilitat administrativa i, així, millorar la seua escalabilitat general.

13. L'objectiu principal dels servidors de configuració en MongoDB és:

A	Adoptar el paper d'àrbitres quan una rèplica falle. Fals. El paper d'àrbitre només pot ser exercit per un servidor “mongod”, mai per un servidor de configuració.

B	Controlar la distribució de dades entre les múltiples particions horitzontals ("shards") existents. Cert. Aquest és el principal objectiu dels servidors de configuració en MongoDB.
C	Respectar el teorema CAP quan es done una partició en la xarxa. Fals. No es requereix la intervenció de cap servidor de configuració en MongoDB quan la xarxa es particione. Aquesta gestió es fa en l'àmbit dels "conjunts de rèpliques" ("*replica set*"). En aquests casos, només el subgrup majoritari podrà continuar i solament quan mantinga més de la meitat de les seues rèpliques.
D	Detectar fallades en els nodes, iniciant un protocol de recuperació quan es done alguna fallada. Fals. La documentació de MongoDB no descriu detalladament els seus mecanismes de detecció de fallades. No està clar quins elements participen en la detecció. No obstant això, aquesta gestió es fa en l'àmbit dels "conjunts de rèpliques", de manera interna a cada conjunt. Això suggereix que la detecció està integrada en les interaccions entre els servidors "mongod" que formen cada conjunt.

14. Considerant la classificació temàtica de vulnerabilitats vista en el Seminari 8, es pot considerar certa la següent afirmació:

A	L'explotació de defectes en polítiques de seguretat no pot automatitzar-se tan fàcilment com l'explotació de contrasenyes febles. Cert. Els defectes en les polítiques de seguretat han de ser acuradament analitzats i comprovats per un atacant per a identificar-los. Això demana molt de temps i interacció amb el sistema. D'altra banda, hi ha múltiples llistes de "contrasenyes febles". Una vegada es conega l'identificador d'un usuari, un atacant pot intentar un accés remot utilitzant alguna d'aquestes llistes. Per tant, l'explotació d'aquesta segona vulnerabilitat pot ser fàcilment automatitzada.
B	El "*phishing*" és una vulnerabilitat de tipus "error de programari". Fals. El "*phishing*" és una vulnerabilitat de tipus "enginyeria social".
C	La protecció física és un exemple de vulnerabilitat d'enginyeria social. Fals. La protecció física és un mecanisme de seguretat. No és una vulnerabilitat.
D	Un defecte en una política de seguretat de protecció personal requereix menor interacció per a ser explotada que un error de programari en el sistema operatiu. Fals. Les vulnerabilitats dels sistemes operatius solen estar documentades (per exemple, en la descripció de les actualitzacions que les corregeixen). Per tant, no es necessita interactuar prèviament amb el sistema per a explotar aquestes vulnerabilitats. Per contra, les polítiques de seguretat de protecció personal no han de ser conegudes per agents externs i els seus defectes, quan n'hi haja, necessitaran major interacció per a ser identificats i explotats en un atac.

15. Assumint aquest Dockerfile…

```
FROM zmq
RUN mkdir /zmq
COPY ./broker.js /zmq/broker.js
WORKDIR /zmq
EXPOSE 8000 8001
CMD node broker.js
```

Quina de les següents afirmacions és FALSA?

A	Necessitem tenir el fitxer "broker.js" en el directori de l'amfitrió en el qual es trobe aquest Dockerfile.

	Sí. La tercera línia del Dockerfile assumeix que el fitxer es troba en el mateix directori del Dockerfile. Els noms de ruta relatius utilitzats com a primer argument de l'ordre COPY assumeixen aquest directori com a origen de la ruta.
B	El programa a executar en aquests contenidors utilitza el port 8000 del contenidor i l'associa al port 8001 de l'amfitrió. Fals. Aquests nombres de port apareixen en la cinquena línia del fitxer. L'ordre EXPOSE llista els nombres de port que va a utilitzar el contenidor per a escoltar connexions. Són ports del contenidor. No s'estableix cap correspondència amb els ports de l'amfitrió.
C	Per omissió, els contenidors generats a partir d'aquest Dockerfile executaran l'ordre "**node broker.js**". Sí. Això s'indica en l'última línia del Dockerfile, utilitzant l'ordre CMD.
D	Aquest Dockerfile assumeix l'existència d'una imatge anomenada "zmq" amb una instal·lació vàlida de l'intèrpret de JavaScript "node". Sí. La primera línia del fitxer indica que la imatge a utilitzar com a base es diu "zmq". L'última línia utilitza l'intèrpret "node". Per tant, es pot assumir que "zmq" ja inclou una instal·lació correcta de l'intèrpret perquè cap de les altres línies del Dockerfile tracta d'instal·lar "node".

16. Imaginem que el component broker de la qüestió 15 s'inclou en un fitxer docker-compose.yml amb aquests continguts (entre uns altres que corresponguen a altres components):

```
version: '2'
services:
...
 bro:
  image: broker
  build: ../broker/
```

Quina de les següents afirmacions és FALSA?

A	Podem iniciar almenys una instància del component **broker** amb l'ordre **docker-compose up –d** Cert. L'ordre "docker-compose up" inicia una instància (o més, si aquestes van ser parades prèviament) de cada component esmentat en **el docker-compose.yml**.

B	Una vegada s'ha iniciat el servei, podem tenir 5 instàncies del component **broker** utilitzant **docker-compose scale broker=5** Fals. El component es diu "**bro**" en aquest fitxer. Per tant, la línia a utilitzar seria: **docker-compose scale bro=5**
C	Aquest fitxer "docker-compose.yml" assumeix que el Dockerfile mostrat en la qüestió 15 es troba en la carpeta "../broker/". Cert. Aquest és l'objectiu de l'ordre "**build**": establir la ubicació del Dockerfile necessari per a construir la imatge.
D	Podem utilitzar l'ordre **docker-compose stop bro** per a parar totes les instàncies d'aquest component **broker**. Cert. Aquesta és l'ordre docker-compose necessària per a fer això.

17. Suposem un protocol de replicació basat en un procés seqüenciador que utilitza un socket PUB ZeroMQ per a propagar tots els esdeveniments "write" als processos participants, en ordre de recepció. Aquests esdeveniments han sigut rebuts mitjançant canals PUSH-PULL, on el seu socket PULL està en el seqüenciador. Aquest protocol de replicació suporta els següents models de consistència:

A	Només el model estricte. Fals. El model de consistència estricte demana un alt grau de coordinació per a ser implantat. Els sockets ZeroMQ suporten comunicació asincrònica. Per això, una implantació senzilla basada en ZeroMQ no pot garantir una consistència estricta.
B	Només el model "*cache*". Fals. La implantació descrita en aquesta qüestió és la vista en el Seminari 5 per a suportar el model de consistència seqüencial. La consistència seqüencial és més forta que la "*cache*". Això significa que els processos també suportaran la consistència "*cache*", però no només aquest model.
C	Només el model causal. Fals. La implantació descrita en aquesta qüestió és la vista en el Seminari 5 per a suportar el model de consistència seqüencial. La consistència seqüencial és més forta que la causal. Això significa que els processos també suportaran la consistència causal, però no només aquest model.
D	Seqüencial, processador, causal, "*cache*" i FIFO. Cert. La implantació descrita en aquesta qüestió és la vista en el Seminari 5 per a suportar el model de consistència seqüencial. La consistència seqüencial és més forta que la processador, causal, "*cache*" i FIFO. Per això, els processos resultants suportaran tots aquests models de consistència.

18. Donada la següent execució:
W1(x)1, R4(x)1, W2(y)2, W1(y)3, W2(x)4, R3(y)2, W3(x)5, R1(x)5, R2(x)1, R3(x)1, R4(y)3, R1(y)2, R3(y)3, R4(x)5, R3(x)4, R2(y)3, R4(y)2, R1(x)4, R2(x)5, R4(x)4.
Aquesta execució suporta aquests models de consistència:

A	Només el model FIFO. Cert. Hi ha cinc accions d'escriptura en aquesta execució. Dos pertanyen a P1 (valor 1 sobre "x" i 3 sobre "y", en aquest ordre), dos més a P2 (valor 2 sobre "y" i 4 sobre "x", en aquest ordre) i una altra a P3 (valor 5 sobre "x"). La consistència FIFO exigeix que els valors escrits per un procés siguen llegits en ordre d'escriptura pels altres processos. Això exigeix que el valor 1 haja de precedir al valor 3 i el valor 2 al valor 4.

	Això s'ha respectat en tots els processos. Aquests dos parells de valors escrits poden ser intercalats entre si lliurement pels lectors. A més, el valor 5 també pot intercalar-se sense restriccions perquè ha sigut escrit per un altre procés. En l'execució, cada procés ha vist aquestes seqüències de valors: P1: 1, 3, 5, 2, 4 P2: 2, 4, 1, 3, 5 P3: 2, 5, 1, 3, 4 P4: 1, 3, 5, 2, 4 Per tant, els processos respecten el model de consistència FIFO. Comprovem ara si es compleixen els altres models de consistència. Començarem amb els models "*cache*" i causal. "*Cache*" no és comparable amb FIFO i és més relaxat que processador i aquest últim más relaxat que seqüencial. Causal és més fort que FIFO i més relaxat que seqüencial. Per tant, si el "*cache*" no se suporta, tampoc se suportarà el processador (ni el seqüencial) i si el causal no se suporta, tampoc se suportarà el seqüencial. La consistència "*cache*" exigeix que tots els processos acorden una seqüència de valors per a cada variable (considerant cada variable per separat). Tenim tres valors per a "x" (1, 4 i 5) i dos per a "y" (2 i 3). Desafortunadament, no hi ha cap acord sobre els valors de "x" (P1 i P4 han vist 1, 5, 4; P2 ha vist 4, 1, 5; i P3 ha vist 5, 1, 4). Això és suficient per a dir que l'execució no respecta el model "*cache*". Per tant, tampoc respectarà el model processador (ni el seqüencial). A part, tampoc hi ha hagut acord sobre els valors de "y" (P1 i P4 han vist 3, 2; mentre P2 i P3 han vist 2, 3). Respecte a la consistència causal, hi ha un "camí" de dependències causals entre els valors 2 i 5 [W2(y)2, …, R3(y)2, W3(x)5,…] ja que l'escriptor del valor 5, P3, va llegir el valor 2 abans d'escriure el valor 5. Això implica que tots els processos han de veure el valor 2 abans que el 5 per a complir amb la consistència causal. Això no ocorre en els processos P1 i P4. Per tant, l'execució no és causal i tampoc serà seqüencial. Per tot això, l'execució només respecta el model FIFO.
B	Només el model "*cache*". Fals. Vegeu l'explicació de l'apartat A.
C	Processador, FIFO i "*cache*". Fals. Vegeu l'explicació de l'apartat A.
D	Seqüencial, processador, causal, "*cache*" i FIFO. Fals. Vegeu l'explicació de l'apartat A.

19. Donat el següent programa servidor de descàrrega de fitxers…

```
var cluster = require('cluster');
var fs = require('fs');
var path = require('path');
var zmq = require('zmq');
var os = require('os');
const ipcName = 'Act2.ipc';
const dlName = 'ipc://'+ipcName;
if (cluster.isMaster) {
   var numCPUs = os.cpus().length;
   var rt = zmq.socket('router');
   var dl = zmq.socket('dealer');
   rt.bindSync('tcp://127.0.0.1:8000');
   dl.bindSync(dlName);
   rt.on('message', function() {
         msg = Array.apply(null, arguments);
         dl.send(msg);   });
   dl.on('message', function() {
         msg = Array.apply(null, arguments);
         rt.send(msg);   });
} else {
  var rep = zmq.socket('rep');
  rep.connect(dlName);
  rep.on('message', function(data) {
    var request = JSON.parse(data);
    fs.readFile(request.path, function(err,
data) {
     if (err) data = ' NOT FOUND';
      rep.send(JSON.stringify({
          pid: process.pid ,
          path: request.path ,
          data: data ,
          timestamp: new Date().toString()
      }))
   })
  })
}
```

Hem tractat d'executar el programa, però no sembla fer res útil. El seu principal problema és…

A	Els sockets ZeroMQ no admeten "**ipc://**" com a transport. Fals. El transport "**ipc:**" pot usar-se per a comunicar dos o més processos situats en un mateix ordinador. Està admès.
B	No s'ha creat cap procés treballador del mòdul "cluster". Cert. Hem de generar múltiples processos treballadors (mitjançant la funció **cluster.fork**) en el codi del procés mestre. No hi ha cap instrucció d'aquest tipus en aquesta versió del programa mostrat en l'enunciat.
C	S'està tractant de propagar missatges interns del mòdul "cluster" a través d'un socket DEALER ZeroMQ. Fals. Els missatges intercanviats entre els processos mestre i treballadors en aquest programa utilitzen canals DEALER-REP de ZeroMQ. No poden qualificar-se com a missatges "interns" del mòdul "cluster". Un canal DEALER-REP en ZeroMQ pot utilitzar-se sense problemes. De fet, és la implantació més senzilla per a comunicar un broker amb diversos treballadors en cas d'utilitzar un patró ROUTER-DEALER en el procés broker.
D	Un servidor no pot utilitzar un socket ROUTER com el seu "endpoint". Fals. Hem vist diversos exemples de serveis replicats que utilitzen un procés broker per a equilibrar la càrrega entre les seues rèpliques. Aquest broker utilitza normalment un socket ROUTER com "endpoint".

20. La vulnerabilitat OpenSSL Heartbleed descrita en el Seminari 8 és un exemple de vulnerabilitat que pertany a les classes següents:

A	"Defecte en la política de treball" en la seua categoria i "Personal humà" a l'origen. Fals. Vegeu la justificació de l'apartat B.

B	“Defecte en la lògica del programari” en la seua categoria i “Biblioteca/middleware” a l'origen. Cert. Aquesta vulnerabilitat va ser causada per un error en la lògica del programari (els missatges de “heartbeat” del protocol SSL necessitaven una resposta d'una grandària especificada pel client i aquesta grandària no estava limitada, podent obtenir així una còpia d'un fragment de la memòria del servidor que podria contenir informació important; per exemple, contrasenyes o certificats) d'una biblioteca (OpenSSL). Per tant, la seua categoria és “defecte en la lògica del programari” i el seu origen és “biblioteca/middleware” perquè el problema afectava aquells programes que estigueren utilitzant les versions vulnerables de la biblioteca OpenSSL.
C	“Enginyeria social” en la seua categoria i “Desenvolupador” a l'origen. Fals. Vegeu la justificació de l'apartat B.
D	“Defecte en la lògica del programari” en la seua categoria i “Personal humà” a l'origen. Fals. Vegeu la justificació de l'apartat B.