UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Faculty of Engineering, Built Environment and Information Technology
Department of Computer Science

Fakulteit Ingenieurswese, Bou-omgewing en Inligtingtegnologie
Departement Rekenaarwetenskap

COS 226: Concurrent Systems

Semester Test 2

Test date:	26 October 2016
Total marks:	60 marks
Time limit:	90 minutes

Name and surname: / *Naam en van:* _____

Student number: / *Studentenommer* _____

Instructions

1. Read the question paper carefully and answer all the questions below. Write your answers on the test paper.
2. This test is comprised of **26** questions on **7** pages. Please check that you have received the entire paper before you begin writing.
3. This is a closed book paper. Switch off your cell phone for the duration of the test. No electronic devices, other than a non-programmable scientific calculator, are allowed. If you are found guilty of cheating, disciplinary action will be taken. Queries will not be accepted for answers written in pencil.

Instruksies

1. *Lees die vraestel noukeurig deur, en beantwoord al die vrae wat volg. Skryf jou antwoorde op die vraestel neer.*
2. *Hierdie vraestel bestaan uit **26** vrae oor **7** bladsye. Maak asseblief seker dat jy die hele vraestel ontvang het voordat jy begin skryf.*
3. *Die vraestel is geslote boek. Skakel jou selfoon af vir die duur van die eksamen. Geen elektroniese toestelle word toegelaat nie, behalwe 'n nie-programmeerbare wetenskaplike sakrekenaar. As jy skuldig bevind word aan oneerlikheid, sal dissiplinêre stappe geneem word. Navrae sal nie aanvaar word vir antwoorde wat in potlood geskryf is nie.*

1. Why does the TTAS lock perform better than the TAS lock? / *Hoekom voer die TTAS slot beter uit as die TAS slot?* [2]

   A. The extra test makes if safer to acquire the lock / *Die ekstra toets maak dit veiliger om die slot te bekom*

   B. TTAS first tests the lock using getAndSet() / *TTAS toets eers die slot deur getAndSet() te gebruik*

   C. The extra test uses get() which results in less traffic / *Die ekstra toets gebruik get() wat minder verkeer tot gevolg het*

   D. TTAS does not create any traffic on the bus / *TTAS skep nie enige verkeer op die bus nie*

2. In the context of the Exponential Backoff Lock, thread backoff occurs when: / *In die konteks van die Eksponensiele Terugstaan Slot, vind thread terugval plaas wanneer:* [2]

   A. there are more than one thread trying to acquire the lock at the same time / *daar meer as een thread is wat op dieselfde tyd die slot probeer bekom*

   B. there are more than one thread in contention / *daar meer as een thread in kontensie is*

   C. there is high contention / *daar hoë kontensie is*

   D. a thread has to wait / *'n thread moet wag*

3. In the context of locking algorithms, a fast-path is: / *In die konteks van slot algoritmes is 'n vinnige-pad:* [2]

   A. a shortcut (through complex mechanisms) for lone threads / *'n kortpad (deur komplekse meganismes) vir enkele threads*

   B. an efficient means for locks to access shared memory / *'n effektiewe manier vir slotte om toegang tot gedeelde geheue te kry*

   C. a path through the lock that allows threads to overtake each other / *'n pad deur die slot wat threads toelaat om mekaar verby te vat*

   D. a path through the lock that reduces contention / *'n pad deur die slot wat kontensie verminder*

4. In terms of a lock, what does contention mean? / *In terme van 'n slot, wat beteken kontensie?* [2]

   A. That only one thread may be in the critical section at a time / *Dat slegs een thread op 'n keer in die kritiese seksie mag wees*

   B. That a thread has to backoff and try again later / *Dat 'n thread moet terugstaan en later weer probeer*

   C. That there are multiple threads trying to acquire the lock at the same time/ *Dat daar veelvuldige threads is wat die slot probeer bekom op dieselfde tyd*

   D. That the lock does not provide fairness / *Dat die slot nie regverdigheid verskaf nie*

5. In a CLH lock, when a thread attempts to acquire the lock ... / *In 'n CLH slot, wanneer 'n thread probeer om die slot te bekom ...* [2]

   A. It sets its own flag to true and spins on its own node / *Sit dit sy eie vlag na waar en spin op sy eie nodus*

   B. It sets its predecessor's flag to true and spins on its own node / *Sit dit sy voorganger se vlag na waar en spin op sy eie nodus*

   C. It sets its successor's flag to true and spins on its predecessors node / *Sit dit sy opvolger se vlag na waar en spin op sy voorganger se nodus*

   D. It sets its own flag to true and spins on its predecessor's node / *Sit dit sy eie vlag na waar en spin op sy voorganger se nodus*

6. The Condition object: / *Die Condition objek*: [2]

   A. provides a prerequisite for when the lock can be acquired / *verskaf 'n voorwaarde vir wanneer 'n slot bekom kan word*

   B. provides the ability to release a lock temporarily / *verskaf die moontlikheid om 'n slot tydelik vry te stel*

   C. is an implementation of the Reentrant Lock / *is 'n implementasie van die Herinskrewe slot*

   D. determines how long a thread can hold a lock before releasing it / *bepaal hoe lank 'n thread 'n slot kan hou voordat hy dit vrystel*

7. The observation that gives rise to a lock that separates threads into readers and writers to improve the performance of the readers is: / *Die waarneming wat tot gevolg het 'n slot wat threads in lesers en skrywers indeel om die werkverrigting van die lesers te verbeter is:* [2]

   A. that readers do not interfere with writers / *dat lesers nie inmeng met skrywers nie*

   B. readers are naturally faster than writers in all situations / *lesers is natuurlik vinniger as skrywers in alle situasies*

   C. readers do not interfere with other readers and writers do not interfere with other writers / *lesers meng nie in met ander lesers en skrywers meng nie in met anders skrywers nie*

   D. readers do not interfere with other readers while writers interfere with other readers and writers / *lesers meng nie in met ander lesers nie terwyl skrywers inmeng met ander lesers en skrywers*

8. A method is lock-free if: / *'n Metode is slotvry as:* [2]

   A. it guarantees that some call always finishes in a finite number of steps / *dit waarborg dat 'n roep altyd sal klaarmaak in 'n eindige hoeveelheid stappe*

   B. it guarantees that every call finishes in a finite number of steps / *dit waarborg dat elke roep sal klaarmaak in 'n eindige hoeveelheid stappe*

   C. its locks are efficient / *sy slotte effektief is*

   D. it contains only one lock / *dit slegs een slot bevat*

9. When a thread blocks ... / *Wanneer 'n thread blokkeer ...* [2]

   A. It sleeps for a certain amount of time and then wakes up to try again / *Slaap dit vir 'n sekere hoeveelheid tyd en word dan wakker om weer te probeer*

   B. It suspends for a certain amount of time and then tries again / *Staak dit vir 'n sekere hoeveelheid tyd en probeer weer*

   C. It keeps testing the lock conditions to determine if it can acquire the lock / *Hou dit aan om die slotkondisies te toets om vas te stel of dit die slot kan bekom*

   D. It acquires the lock and waits until it can continue / *Bekom dit die slot en wag totdat dit kan aangaan*

10. The SimpleReadWriteLock is not fair, because ... / *Die SimpleReadWrite slot is nie regverdig nie, want ...* [2]

    A. The writer could starve / *Die skrywer kan uithonger*

    B. The reader could starve / *Die leser kan uithonger*

    C. There are more readers than writers / *Daar is meer lesers as skrywers*

    D. Readers can read at the same time / *Lesers kan op dieselfde tyd lees*

11. Fine-grained synchronization performs better than Optimistic synchronization when: / *Fyngrein sinkronisasie voer beter uit as Optimistiese sinkronisasie wanneer:* [2]

    A. The cost of validation is higher than the cost of locking individual nodes / *Die kostes van validasie is meer as die kostes van om individuele nodusse te sluit*

    B. Locking individual nodes are not efficient / *Dit nie effektief is om individuele nodusse te sluit nie*

    C. The cost of compareAndSet() operations are higher than the cost of validation / *Die kostes van compareAndSet() bewerkings meer is as die kostes van validasie*

    D. None of the above, fine-grained synchronization will never perform better than optimistic synchronization / *Nie een van die bogenoemde nie, fyngrein sinkronisasie sal nooit beter uitvoer as optimistiese sinkronisasie nie*

12. In the context of a linked list data structure, optimistic synchronization is so called because: / *In die konteks van 'n geskakelde lys datastruktuur, word optimistiese sinkronisasie so genoem omdat:* [2]

    A. threads that fail to acquire the necessary lock will time out and try to acquire it again / *'n thread wat misluk om die nodige slot te bekom sal uit tyd uit hardloop en weer probeer om dit te bekom*

    B. threads will never give up in trying to acquire the necessary lock so will never time out / *threads sal nooit opgee om te probeer om die nodige slot te bekom nie so hulle sal nooit uit tyd uithardloop nie*

    C. threads are not synchronized by means of any locks / *threads word nie deur middel van enige slotte gesinkroniseer nie*

    D. threads traverse the nodes of the list without locking them, lock the nodes they will work with and then check that those nodes did not change / *threads beweeg deur die nodusse van die lys sonder om hulle te sluit, sluit dan die nodusse waarmee hulle gaan werk en maak dan seker dat daardie nodusse nie verander het nie*

13. Which of the following is an advantage of coarse-grained synchronization? / *Watter van die volgende is 'n voordeel van growwe-grein sinkronisasie?* [2]

    A. Hand-over-hand locking ensures that threads cannot interfere with one another / *Hand-oor-hand sluiting verseker dat threads nie met mekaar kan inmeng nie*

    B. Coarse-grained synchronization is lock-free / *Growwe-grein sinkronisasie is slotvry*

    C. During validation you only need to check the boolean field to determine if the node is still part of the list / *Gedurende validasie hoef jy net die boolean veld te toets om te bepaal of die nodus nog deel is van die lys*

    D. Coarse-grained synchronization is guaranteed to be correct / *Growwe-grein sinkronisasie is gewaarborg om korrek te wees*

14. In the context of a linked list data structure, optimistic synchronization subjects threads to potential: / *In die konteks van 'n geskakelde lys datastruktuur, kan optimistiese sinkronisase lei tot potensiale:* [2]

    A. starvation / *uithongering*

    B. deadlock / *dooiepunt*

    C. interference / *inmenging*

    D. contention / *kontensie*

15. Non-blocking synchronization guarantees progress, but at what cost? / *Nie-blokkerende sinkronisasie waarborg vooruitgang, maar teen watter koste?* [2]

    A. It uses expensive atomic operations / *Dit gebruik duur atomiese bewerkings*

B. It does not use a lock, so there is no mutual exclusion / *Dit gebruik nie 'n slot nie, so daar is nie wedersydse uitsluiting nie*

C. It does not use memory barriers, so results are not guaranteed / *Dit gebruik nie geheueversperrings nie, so resultate is nie gewaardborg nie*

D. It uses expensive hand-over-hand locking / *Dit gebruik duur hand-oor-hand sluiting*

16. What is meant by the statement, "for obvious reasons, spinning makes sense only on multiprocessors"? / *Wat word bedoel met die stelling, "vir duidelike redes maak dit sin om te spin net op multiverwerkers"?* [2]

Solution: In a uniprocessor system, the spinning thread will in any case be suspended [1] when the operating system scheduler switches to running another thread [1] OR a thread will spin until another thread changes the condition that it is waiting for - on a uniprocessor the spinning thread will be the only thread executing at a specific time so another thread will not be able to change the condition [2]

17. 17.1 Give one disadvantage of an Exponential Backoff Lock over a Queue lock. / *Gee een nadeel van 'n Exponensiele Terugstaan slot bo 'n Touslot.* [1]

Solution: Possible critical section underutilization OR cache-coherence traffic OR a backoff lock does not provide fairness

17.2 Give one advantage of an Exponential Backoff Lock over a Queue lock. / *Gee een voordeel van 'n Eksponensiele Terugstaan slot bo 'n Touslot.* [1]

Solution: For an exponential backoff lock it is trivial for a thread to timeout. //Don't give marks for "easier to implement", also don't give marks for "more space efficient" unless the student gives a very good justification

18. What is the main advantage of linked list-based queue locks over array-based queue locks? / *Wat is die hoof voordeel van geskakelde lys-gebaseerde touslotte oor skikking-gebaseerde touslotte?* [2]

Solution: Queue-based locks are more space efficient [1] OR you don't have to worry about creating a circular array since the linked list is not bounded

19. In the MCS Queue lock, why does a thread need to double-check for a successor when unlocking? / *In die MCS Touslot, hoekom moet 'n thread tweekeer kyk of hy 'n opvolger het wanneer oopgesluit word?* [3]

Solution: Threads signal their successors when they unlock so if there is a successor, but the thread did not signal it then that thread will not access the critical section and the queue lock will deadlock OR because another thread could be in the process of being added as the thread's successor but the next field has not been activated yet, so the current thread will not know about the successor
Important concepts:
- Thread has to signal successor[1]
- Thread might not know about successor when unlocking[1] - If there is a successor that was not signalled upon unlocking then deadlock will occur[1]

20. How can a CLH queue lock be modified to be able to deal with timeouts? Briefly explain the functioning of an abortable CLH lock. / *Hoe kan 'n CLH touslot aangepas word om tydsverstreking te kan hanteer? Verduidelik kortliks die funksionering van 'n stakende CLH slot.* [3]

Solution: When a thread times out it marks its node as aborted [1]. The successor in the queue (that was spinning on its predecessor the aborted node) notices that the node has been abandoned [1] and starts spinning on the predecessor of the aborted node [1].

21. Why must a condition be associated with a lock? / *Hoekom moet 'n kondisie met n slot geassosieer word?* [2]

Solution: So that the lock may be released [1] when the thread currently holding it waits on the condition [1]

22. Consider the code given below. / *Bestudeer die onderstaande kode.*

```
1  //...
2  try {
3    if (!property)
4      condition.await();
5  } catch (InterruptedException e) {
6    // Handle interrupt...
7  }
8  //...
```

22.1 In line 4 a thread blocks on a specific condition. Name two ways in which a blocking thread can be woken up. / *In lyn 4 blokkeer 'n thread op 'n spesifieke kondisie. Noem twee maniere waarop 'n blokkerende thread wakker gemaak kan word.* [2]

Solution:

1. By means of a signal via signal(), notify(), signalAll(), or notifyAll() //Any one or all of these are fine.
2. By means of a timeout that is set when a thread waits on a condition.

22.2 The above code has a flaw. Explain the flaw and how it can be fixed. / *Die kode hierbo het 'n fout. Verduidelik die fout en how dit reggemaak kan word.* [3]

Solution: The if-statement in line 3 should be replaced with a while-loop [1].
The property might have become false after it became true and before a waiting thread was signalled. [1] The signalled thread would then not check the property again and would continue to execute while the object is in an inconsistent state. [1]
[Alternatively...]
A thread may wake up for no reason or many threads could have been woken up [1] and pass through even though the condition is not favourable [1].

23. Assume you have a bounded FIFO queue that is implemented as a Producer/Consumer problem. Study the following steps that have already taken place and complete the steps to indicate how lost wakeups can occur. / *Aanvaar jy het 'n gebonde FIFO tou wat geimplementeer is as 'n Produseerder/Verbruiker probleem. Bestudeer die volgende stappe wat klaar plaasgevind het en voltooi die stappe om aan te dui hoe verlore wakker word kan gebeur.* [3]

Assume a full queue. / *Aanvaar 'n vol tou.*

1. Thread A tries to enqueue an item, finds that the queue is full and blocks. / *Thread A probeer om 'n item by te voeg, vind uit dat die tou vol is en blokkeer.*

2. Thread B tries to enqueue an item, finds that the queue is full and blocks. / *Thread B probeer om 'n item by te voeg, vind uit dat die tou vol is en blokkeer.*
3. Thread C dequeues an item from the queue and signals thread A that it can wake up and continue. / *Thread C haal 'n item van die tou af en sein vir Thread A dat dit kan wakker word en aangaan.*

...

Solution:

24. How does fine-grained synchronization improve upon coarse-grained synchronization? / *Hoe verbeter fyn-grein sinkronisasie op growwe-grein sinkronisasie?* [2]

Solution: Since each node now has an individual lock[1], threads can access different areas of the linked list at the same time [1]

25. The contains() method of lazy synchronization is more efficient that the contains() method of optimistic synchronization. Why? / *Die contains() metode van lui sinkronisasie is meer doeltreffend as die contains() metode van optimistiese sinkronisasie. Hoekom?* [2]

Solution: The contains method of lazy synchronization is wait-free [1]. It only checks the value of the marked field to determine if the element is still in the list, while with optimistic the whole list retraversed [1].

26. Imagine that you are writing an application and have the option of using either coarse-grained synchronization or non-blocking synchronization to ensure the safety of a linked list that will be accessed by multiple threads. Which synchronization is more appropriate for the task? Describe the implications of choosing each synchronization. / *Verbeel jou dat jy 'n toepassing skryf en die opsie het van om of growwe-grein sinkronisasie of nie-blokkerende sinkronisasie te gebruik om die veiligheid van 'n geskakelde lys wat deur vele threads gebruik sal word te verseker. Watter sinkronisasie is meer gepas vir die taak? Beskryf die implikasies van om elk van die sinkronisasies te kies.* [4]

Solution: A coarse-grained mechanism will be easier (quicker) to implement [1] and will perform well if contention for the data structure is low. [1]

A non-blocking synchronization mechanism will be more difficult (slower) to implement [1] but will outperform the coarse-grained mechanism in the presence of high contention for the data structure. [1]

[A student may alternatively comment on the fact that a coarse-grained synchronization mechanism is more efficient for single threads to execute [1], while the overhead of executing a non-blocking synchronization mechanism makes such a mechanism worthwhile only to synchronize many threads. [1]]