Dreadlocks

Konrad Błachnio
K.Blachnio@students.mimuw.edu.pl

MIMUW
19 maja 2010

Co to jest lock manager?

Wikipedia:

A distributed lock manager (DLM)
provides distributed applications with a means to synchronize their accesses to shared resources.

Co to jest lock manager?

Wikipedia:

A distributed lock manager (DLM)
provides distributed applications with a means to synchronize their accesses to shared resources.

Czyli:

Moduł zarządzający zamkami
zapewniający rozproszonym aplikacjom możliwość synchronizowania dostępu do współdzielonych zasobów.

Lock Manager - cechy

Specyfika zarządcy zamków

- Jest jeden
- Musi obsługiwać wiele (dziesiątek) procesów
- Musi obsługiwać wiele (milionów) zamków

Lock Manager - cechy

Specyfika zarządcy zamków

- Jest jeden
- Musi obsługiwać wiele (dziesiątek) procesów
- Musi obsługiwać wiele (milionów) zamków

Pożądane cechy zarządcy zamków

- Bezbłędność
- Współbieżna obsługa tak dużej ilości procesów jak to możliwe
- Duża szybkość działania
- Niewielki rozmiar
- Racjonalne zarządzanie zasobami

Lock manager wizualizacja

Zamki - typy

Typ dostępu do zasobu (obiektu)

- None Lock (NL) czyli żaden

Zamki - typy

Typ dostępu do zasobu (obiektu)

- None Lock (NL) czyli żaden
- Concurrent Read (CR) czyli współbieżne czytanie

Zamki - typy

Typ dostępu do zasobu (obiektu)

- None Lock (NL) czyli żaden
- Concurrent Read (CR) czyli współbieżne czytanie
- Concurrent Write (CW) czyli współbieżne pisanie oraz czytanie

Zamki - typy

Typ dostępu do zasobu (obiektu)

- None Lock (NL) czyli żaden
- Concurrent Read (CR) czyli współbieżne czytanie
- Concurrent Write (CW) czyli współbieżne pisanie oraz czytanie
- Protected Read (PR) czyli wyłączne czytanie

Zamki - typy

Typ dostępu do zasobu (obiektu)

- None Lock (NL) czyli żaden
- Concurrent Read (CR) czyli współbieżne czytanie
- Concurrent Write (CW) czyli współbieżne pisanie oraz czytanie
- Protected Read (PR) czyli wyłączne czytanie
- Protected Write (PW) czyli wyłączne pisanie

Zamki - typy

Typ dostępu do zasobu (obiektu)

- None Lock (NL) czyli żaden
- Concurrent Read (CR) czyli współbieżne czytanie
- Concurrent Write (CW) czyli współbieżne pisanie oraz czytanie
- Protected Read (PR) czyli wyłączne czytanie
- Protected Write (PW) czyli wyłączne pisanie
- Exclusive (EX) czyli wyłączne wszystko

Zamki - tabela dostępu

Mode	NL	CR	CW	PR	PW	EX
NL	Yes	Yes	Yes	Yes	Yes	Yes
CR	Yes	Yes	Yes	Yes	Yes	No
CW	Yes	Yes	Yes	No	No	No
PR	Yes	Yes	No	Yes	No	No
PW	Yes	Yes	No	No	No	No
EX	Yes	No	No	No	No	No

Zamek - cechy

Specyfika zamka

- Jest wiele zamków na raz (jeden dla każdego obiektu)
- Musi obsługiwać wiele (kilka) procesów

Zamek - cechy

Specyfika zamka

- Jest wiele zamków na raz (jeden dla każdego obiektu)
- Musi obsługiwać wiele (kilka) procesów

Pożądane cechy zamków

- Chcemy aby były małe
- Chcemy aby obsługiwały wszystkie poziomy
- Powinien działać tak szybko jak to możliwe

Dwa poziomy problemu

Organizacja (implementacja) zamka

- Aby był możliwie mały
- Aby wspierał wszystkie poziomy
- Aby był maksymalnie współbieżny

Dwa poziomy problemu

Organizacja (implementacja) zamka

- Aby był możliwie mały
- Aby wspierał wszystkie poziomy
- Aby był maksymalnie współbieżny

Organizacja (implementacja) zarządcy

- Aby działał poprawnie
- Aby był maksymalnie współbieżny
- Aby był maksymalnie szybki
- Aby był maksymalnie mały

Application Programming Interface

API (interfejs programowania aplikacji)

ILockManager

Application Programming Interface

API (interfejs programowania aplikacji)

ILockManager
ILockManagerWithOwnerTracing<OWNER>

Application Programming Interface

API (interfejs programowania aplikacji)

ILockManager
ILockManagerWithOwnerTracing<OWNER>
ILockManagerWithDeadlockDetecting<OWNER>

ILockManager - API

Przegląd dostępnych funkcji i parametry

void getLock(long objId, Type lockType)
void releaseLock(long objId, Type LockType)

ILockManager - API

Przegląd dostępnych funkcji i parametry

void getLock(long objId, Type lockType)
void releaseLock(long objId, Type LockType)
void promoteLock(long objId, Type lockType, Type newType)
void demoteLock(long objId, Type lockType, Type newType)

ILockManager - API

Przegląd dostępnych funkcji i parametry

void getLock(long objId, Type lockType)
void releaseLock(long objId, Type LockType)
void promoteLock(long objId, Type lockType, Type newType)
void demoteLock(long objId, Type lockType, Type newType)
void changeLock(long objId, Type lockType, Type newType)

ILockManagerWithOwnerTracing<OWNER> - API

Przegląd dostępnych funkcji i parametry

void getLock(OWNER owner, long objectId, Type newType)
void releaseLock(OWNER owner,long objectId)
void promoteLock(OWNER owner,long objectId, Type newType)
void demoteLock(OWNER owner,long objectId, Type newType)
void changeLock(OWNER owner,long objectId, Type newType)

ILockManagerWithOwnerTracing<OWNER> - API

Przegląd dostępnych funkcji i parametry

void getLock(OWNER owner, long objectId, Type newType)
void releaseLock(OWNER owner,long objectId)
void promoteLock(OWNER owner,long objectId, Type newType)
void demoteLock(OWNER owner,long objectId, Type newType)
void changeLock(OWNER owner,long objectId, Type newType)
Type checkCurrentLockType(OWNER owner, long objectId)

ILockManagerWithOwnerTracing<OWNER> - API

Przegląd dostępnych funkcji i parametry

void getLock(OWNER owner, long objectId, Type newType)
void releaseLock(OWNER owner,long objectId)
void promoteLock(OWNER owner,long objectId, Type newType)
void demoteLock(OWNER owner,long objectId, Type newType)
void changeLock(OWNER owner,long objectId, Type newType)
Type checkCurrentLockType(OWNER owner, long objectId)
void interruptCurrentCalls(OWNER owner)
void releaseAllLocksOwnedBy(OWNER owner)

Pomysł na implementację

Bierzemy współbieżną strukturę danych

Pomysł na implementację

Bierzemy współbieżną strukturę danych

ConcurrentHashMap
Hashtable

Pomysł na implementację

Bierzemy współbieżną strukturę danych

ConcurrentHashMap
Hashtable

Sami implementujemy współbieżną strukturę danych

Pomysł na implementację

Bierzemy współbieżną strukturę danych

ConcurrentHashMap
Hashtable

Sami implementujemy współbieżną strukturę danych

Lock
Synchronized

getLock

```
...

if (data.containsKey(objectId))
  objectLock = data.get(objectId);
else {
  objectLock = new Lock();
  data.put(objectId, objectLock);
}

objectLock.getLockFor(objectId, lockType);

...
```

releaseLock

```
...

objectLock.release(objectId, lockType);

if (objectLock.isEmpty)
  data.remove(objectId);

...
```

Anomalie danych

Wyścig procesów

- getLock i getLock

Anomalie danych

Wyścig procesów

- getLock i getLock
- getLock i releaseLock

Anomalie danych

Wyścig procesów

- getLock i getLock
- getLock i releaseLock
- promoteLock i releaseLock

Anomalie danych

Wyścig procesów

- getLock i getLock
- getLock i releaseLock
- promoteLock i releaseLock

Dlaczego te są w porządku?

- promoteLock i promoteLock

Anomalie danych

Wyścig procesów

- getLock i getLock
- getLock i releaseLock
- promoteLock i releaseLock

Dlaczego te są w porządku?

- promoteLock i promoteLock
- promoteLock i demoteLock

Anomalie danych

Wyścig procesów

- getLock i getLock
- getLock i releaseLock
- promoteLock i releaseLock

Dlaczego te są w porządku?

- promoteLock i promoteLock
- promoteLock i demoteLock
- demoteLock i releaseLock

Naprawa anomalii

Atomowe operacje

- getLock w stosunku do getLock

Naprawa anomalii

Atomowe operacje

- getLock w stosunku do getLock
- getLock i promoteLock w stosunku do releaseLock

ILockManagerWithDeadlockDetecting<OWNER> - API

Przegląd dostępnych funkcji i parametry

void getLock(OWNER owner, long objectId, Type newType)
void releaseLock(OWNER owner,long objectId)
void promoteLock(OWNER owner,long objectId, Type newType)
void demoteLock(OWNER owner,long objectId, Type newType)
void changeLock(OWNER owner,long objectId, Type newType)
Type checkCurrentLockType(OWNER owner, long objectId)
void interruptCurrentCalls(OWNER owner)
void releaseAllLocksOwnedBy(OWNER owner)

ILockManagerWithDeadlockDetecting<OWNER> - API

Przegląd dostępnych funkcji i parametry

void getLock(OWNER owner, long objectId, Type newType)
void releaseLock(OWNER owner,long objectId)
void promoteLock(OWNER owner,long objectId, Type newType)
void demoteLock(OWNER owner,long objectId, Type newType)
void changeLock(OWNER owner,long objectId, Type newType)
Type checkCurrentLockType(OWNER owner, long objectId)
void interruptCurrentCalls(OWNER owner)
void releaseAllLocksOwnedBy(OWNER owner)
WaitingForLockOn<OWNER>
public OWNER getOwner()
public long getWaitingForLockOn()

ILockManagerWithDeadlockDetecting<OWNER> - API

Przegląd dostępnych funkcji i parametry

void getLock(OWNER owner, long objectId, Type newType)
void releaseLock(OWNER owner,long objectId)
void promoteLock(OWNER owner,long objectId, Type newType)
void demoteLock(OWNER owner,long objectId, Type newType)
void changeLock(OWNER owner,long objectId, Type newType)
Type checkCurrentLockType(OWNER owner, long objectId)
void interruptCurrentCalls(OWNER owner)
void releaseAllLocksOwnedBy(OWNER owner)
WaitingForLockOn<OWNER>
public OWNER getOwner()
public long getWaitingForLockOn()
void setOnDeadlockCall-
back(Callback<List<WaitingForLockOn<OWNER»,
RuntimeException> deadlockCallback)

Deadlock - zakleszczenie, definicja

Wikipedia:

A deadlock is a situation when two or more processes are each waiting for each other to release a resource.

Deadlock - zakleszczenie, definicja

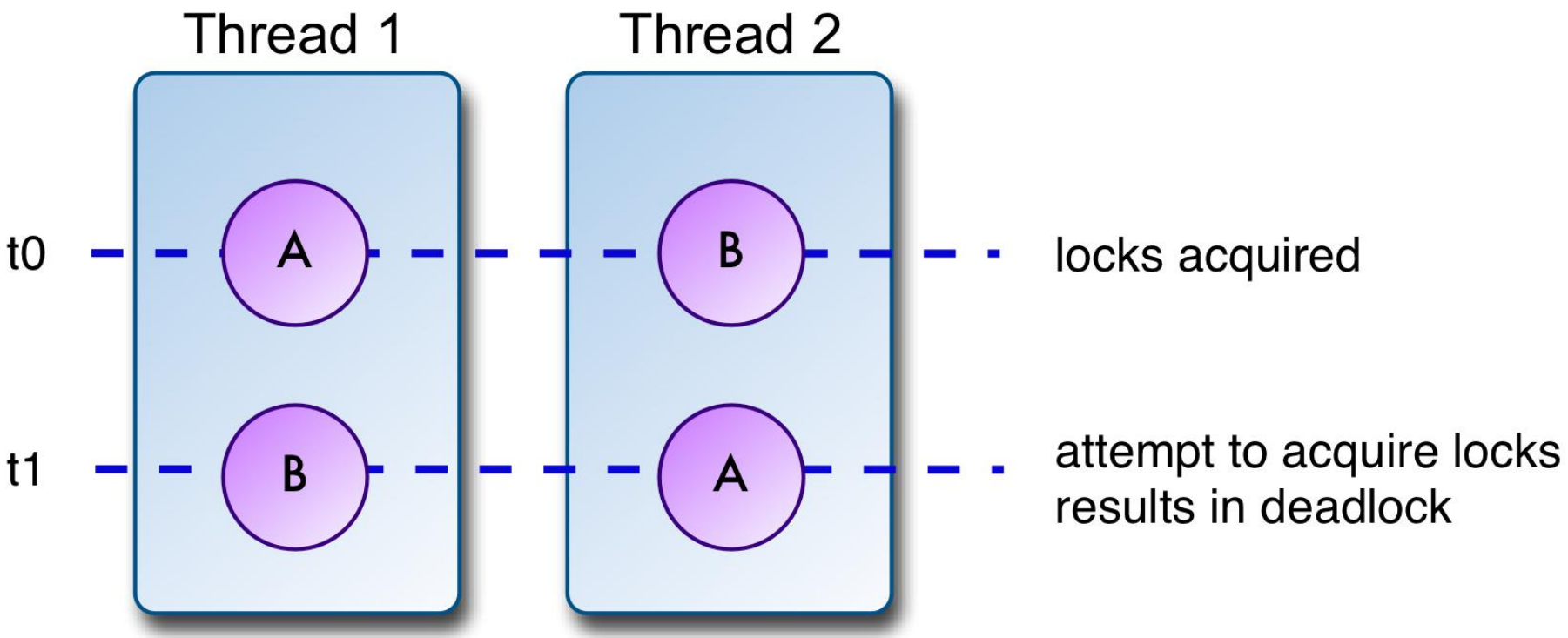
Wikipedia:

A deadlock is a situation when two or more processes are each waiting for each other to release a resource.

Czyli:

Do zakleszczenia dochodzi gdy dwa, lub więcej procesów, czekają wzajemnie na zwolnienie zasobu trzymanego przez drugi proces.

Deadlock - przykład



Deadlock - metody radzenia sobie z problemem

Metoda 1 - Ignorujemy

Deadlock - metody radzenia sobie z problemem

Metoda 1 - Ignorujemy

Zalety:

- Banalna implementacja

Deadlock - metody radzenia sobie z problemem

Metoda 1 - Ignorujemy

Zalety:
- Banalna implementacja

Wady:
- Cała odpowiedzialność na głowie użytkownika
- Zawieszanie się aplikacji

Deadlock - przykład

Deadlock - metody radzenia sobie z problemem

Metoda 2 - Wprowadzamy timeout

Deadlock - metody radzenia sobie z problemem

Metoda 2 - Wprowadzamy timeout

Zalety:

- Prostota implementacji
- Użytkownik nie musi dużo robić

Deadlock - metody radzenia sobie z problemem

Metoda 2 - Wprowadzamy timeout

Zalety:
- Prostota implementacji
- Użytkownik nie musi dużo robić

Wady:
- Wartość timeoutu

Deadlock - metody radzenia sobie z problemem

Metoda 3 - Dajemy użytkownikowi możliwość interrupt

Deadlock - metody radzenia sobie z problemem

Metoda 3 - Dajemy użytkownikowi możliwość interrupt

Zalety:

- Pozostawiamy użytkownikowi dużą dozę swobody

Deadlock - metody radzenia sobie z problemem

Metoda 3 - Dajemy użytkownikowi możliwość interrupt

Zalety:

- Pozostawiamy użytkownikowi dużą dozę swobody

Wady:

- Pozostawiamy użytkownikowi dużą dozę swobody

Deadlock - przykład

Deadlock - metody radzenia sobie z problemem

Metoda 4 - Błąkacz czyli timeout i interrupt

Deadlock - metody radzenia sobie z problemem

Metoda 4 - Błąkacz czyli timeout i interrupt

Zalety:

- Użytkownik nie musi nic robić

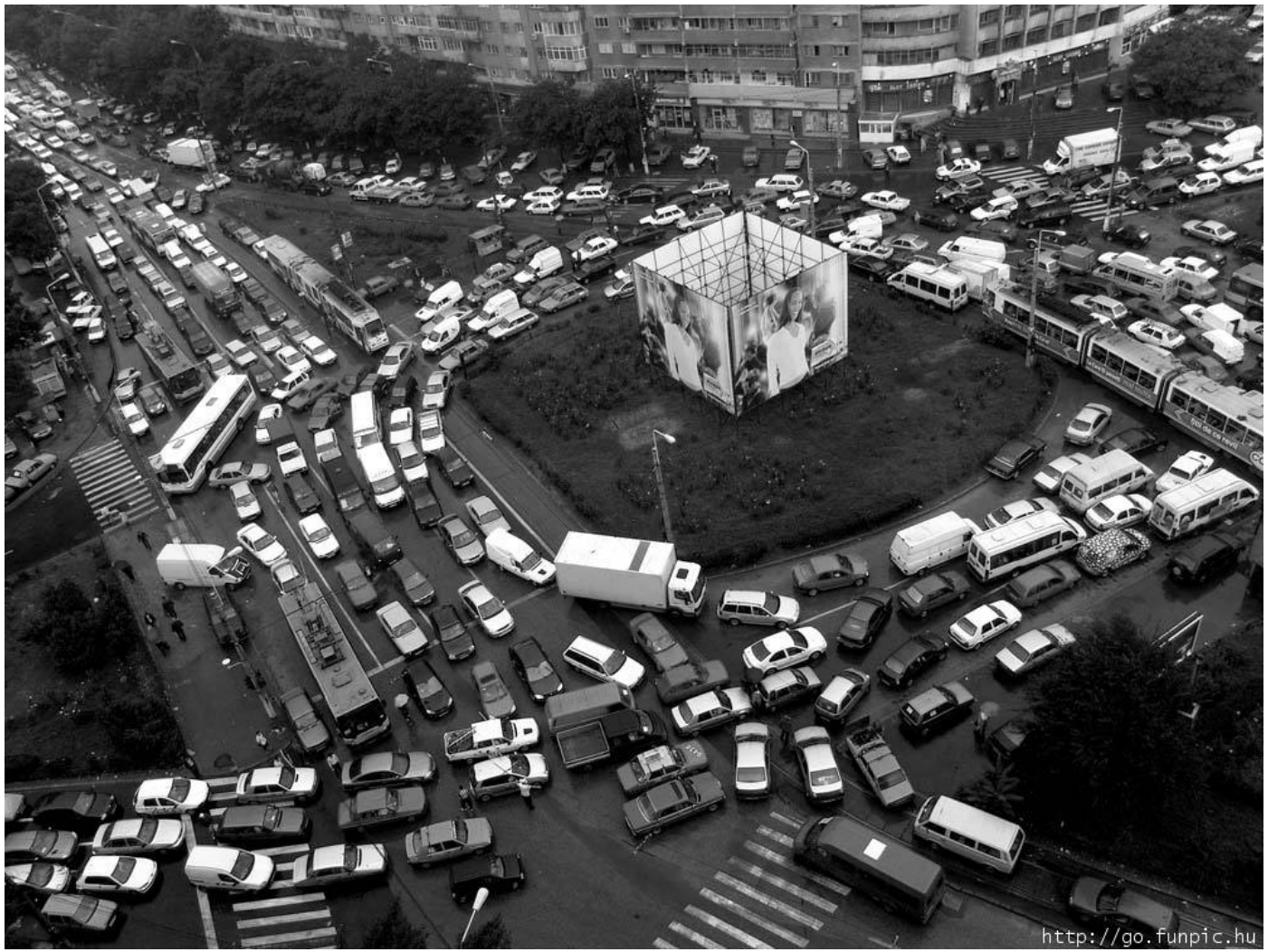
Deadlock - metody radzenia sobie z problemem

Metoda 4 - Błąkacz czyli timeout i interrupt

Zalety:
- Użytkownik nie musi nic robić

Wady:
- Wartość timeoutu
- Skomplikowana implementacja
- Które transakcje zwijać

Deadlock - przykład



Deadlock - metody radzenia sobie z problemem

Metoda 5 - Branie zamków w ustalonej kolejności

Deadlock - metody radzenia sobie z problemem

Metoda 5 - Branie zamków w ustalonej kolejności

Zalety:

- Poprawność

Deadlock - metody radzenia sobie z problemem

Metoda 5 - Branie zamków w ustalonej kolejności

Zalety:
- Poprawność

Wady:
- Skomplikowana implementacja
- Dziwne API

Deadlock - metody radzenia sobie z problemem

Metoda 5 - Branie zamków w ustalonej kolejności

Zalety:
- Poprawność

Wady:
- Skomplikowana implementacja
- Dziwne API

Metoda 6 Dreadlocks - straszny zamek

`www.cl.cam.ac.uk/~ejk39/papers/dreadlocks-spaa08.pdf`

Dreadlocks - straszny zamek

Dreadlocks - straszny zamek

Pomysł:

- Wykrywać potencjalne zakleszczenia zanim one zajdą
- Nie pozwalać na wzięcie zamka jeśli prowadzi to do zakleszczenia

Dreadlocks - straszny zamek

Pomysł:

- Wykrywać potencjalne zakleszczenia zanim one zajdą
- Nie pozwalać na wzięcie zamka jeśli prowadzi to do zakleszczenia

Zalety:

- Prosty pomysł
- Użytkownik nie musi dużo robić

Dreadlocks - straszny zamek

Pomysł:

- Wykrywać potencjalne zakleszczenia zanim one zajdą
- Nie pozwalać na wzięcie zamka jeśli prowadzi to do zakleszczenia

Zalety:

- Prosty pomysł
- Użytkownik nie musi dużo robić

Wady:

- Dodatkowo przechowywane dane
- Dodatkowo wykonywane operacje

Dreadlocks - jak to wygląda

Założenia:

- Każdy wątek wie na kogo czeka
- Każdy zamek wie kto jest jego właścicielem

Dreadlocks - jak to wygląda

Założenia:

- Każdy wątek wie na kogo czeka
- Każdy zamek wie kto jest jego właścicielem

Nie próbuję zająć żadnego zamka

- Czekam tylko na siebie

Dreadlocks - jak to wygląda

Założenia:

- Każdy wątek wie na kogo czeka
- Każdy zamek wie kto jest jego właścicielem

Nie próbuję zająć żadnego zamka

- Czekam tylko na siebie

Próbuję zająć zamek

- Czekam na siebie
- Czekam na wszystkich na których czeka właściciel zamka

Dreadlocks - jak to wygląda

Założenia:

- Każdy wątek wie na kogo czeka
- Każdy zamek wie kto jest jego właścicielem

Nie próbuję zająć żadnego zamka

- Czekam tylko na siebie

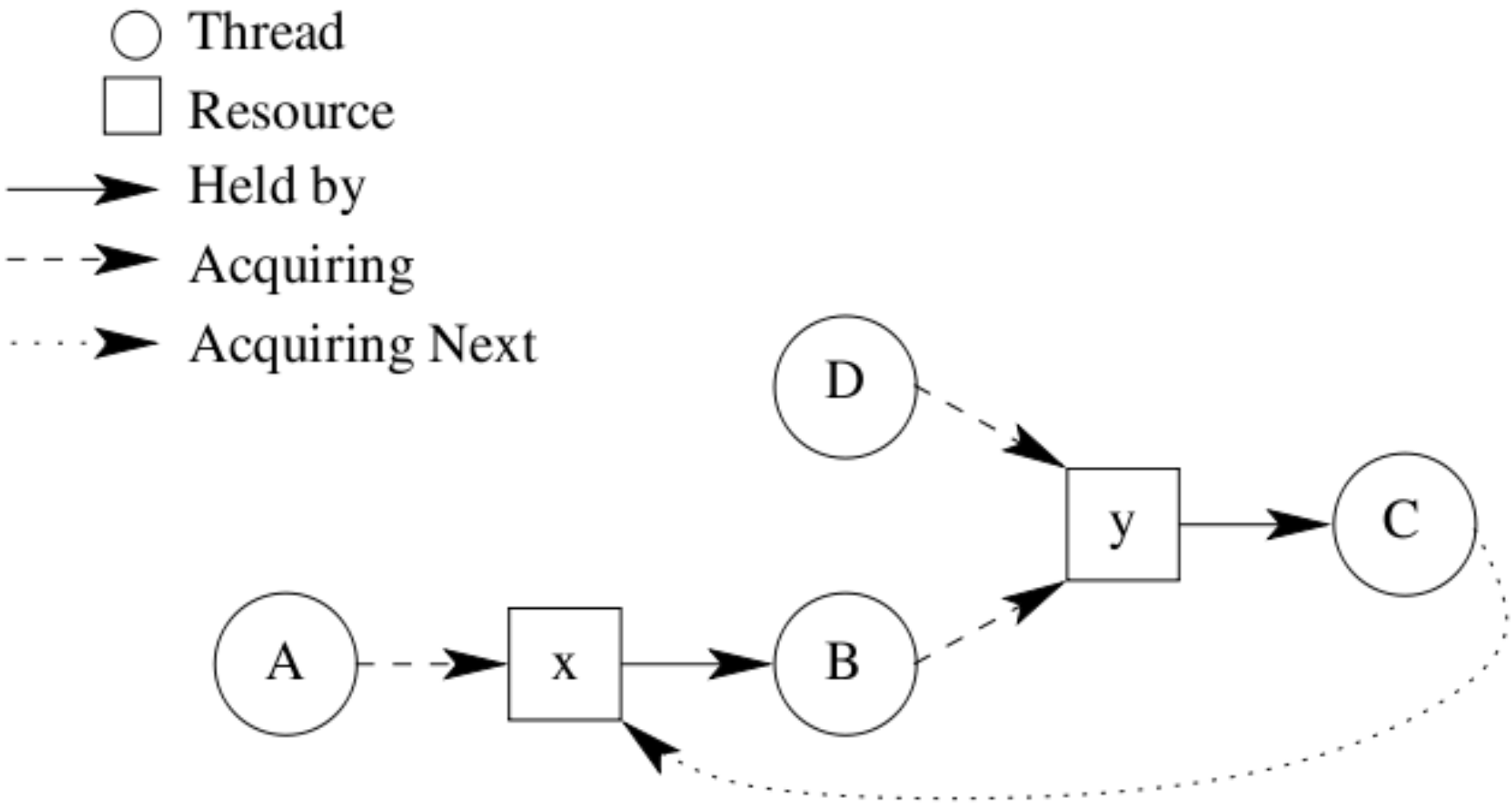
Próbuję zająć zamek

- Czekam na siebie
- Czekam na wszystkich na których czeka właściciel zamka

Zakleszczenie występuje wtedy gdy

- Właściciel zamka czeka na mnie

Dreadlocks - straszny zamek - przykład



Dreadlocks - straszny zamek - przykład cd.

Na który wątek czekają wątki z przykładu:

```
C = {C}
D = {C,D}
B = {C,B}
A = {C,B,A}
```

Dreadlocks - straszny zamek - przykład cd.

Na który wątek czekają wątki z przykładu:

```
C = {C}
D = {C,D}
B = {C,B}
A = {C,B,A}
```

Wątek C próbuje zająć zamek x

Dreadlocks - straszny zamek - przykład cd.

Na który wątek czekają wątki z przykładu:

```
C = {C}
D = {C,D}
B = {C,B}
A = {C,B,A}
```

Wątek C próbuje zająć zamek x
Wątek B właściciel zamka x czeka na wątki {C,B}

Dreadlocks - straszny zamek - przykład cd.

Na który wątek czekają wątki z przykładu:

```
C = {C}
D = {C,D}
B = {C,B}
A = {C,B,A}
```

Wątek C próbuje zająć zamek x
Wątek B właściciel zamka x czeka na wątki {C,B}
Zakleszczenie

Źródła

Dreadlocks:
www.cl.cam.ac.uk/~ejk39/papers/dreadlocks-spaa08.pdf

Lock Manager:
http://en.wikipedia.org/wiki/Distributed_lock_manager

Deadlock:
http://en.wikipedia.org/wiki/Deadlock

Dziękuję

Dziękuję

Dziękuję

Dziękuję

Starczy czasu?

Obecna implementacja

Dlaczego nie jest dobrze

Brzydki kod

Brak komentarzy

Skomplikowana implementacja?

Obecna implementacja

Dlaczego nie jest dobrze

Brzydki kod
Brak komentarzy
Skomplikowana implementacja?

Dlaczego jest źle

Błędy w implementacji

Obecna implementacja

Dlaczego nie jest dobrze

Brzydki kod
Brak komentarzy
Skomplikowana implementacja?

Dlaczego jest źle

Błędy w implementacji

Dlaczego jest bardzo źle

Błędy w założeniach

Lock vs. Synchronized

- Szybszy jest Lock dla większej ilości procesów
- Szybszy jest Synchronized dla mniejszej ilości procesów

Hashtable vs. ConcurrentHashMap

- Szybszy jest ConcurrentHashMap dla większej ilości procesów
- Szybszy jest Hashtable dla mniejszej ilości procesów