# Betriebssysteme Übung
# 2. Tutorium

# Task 1 – Locks (1)

*Wozu Locks?*

- Dienen dazu, exklusiven Zugriff auf eine Ressource sicherzustellen
- Lock = binäre Semaphore
- Wenn ein Thread einen besetzten Lock anfordert:

  Thread->Sleep()

  was passiert dabei im Sheduler?

# Task 1 – Locks (2)

*Bestandteile:*

- Variable mit Thread*, der den Lock hält
- Liste der Thread*, die den Lock angefordert haben
- Lock::Acquire():

  anfordern des Lock (atomar!)
- Lock::Release():

  freigeben des Lock (atomar!)
- Lock::isHeldByCurrentThread():

  Servicemethode, die *true* liefert, wenn *currentThread* den Lock hält.

# Task 1 – Locks (3)

*Mehrere Implementierungsvarianten*

- Ableitung von Semaphore
- ohne Rekursion

  Mehrfacher Aufruf von Acquire() darf nicht zu Deadlock führen,

  mehrfacher Aufruf von Release() ohne Auswirkungen
- mit Rekursion

  Mehrfacher Aufruf von Acquire() möglich,

  tatsächliches Release() erst nach gleicher Anzahl

# Task 1 – Conditions (1)

*Wozu Condition Variablen?*

- Prozesssynchronisation
- verwenden dazu einen Lock

# Task 1 – Conditions (2)

*Bestandteile:*

- Liste der wartenden Thread*
- Condition::Wait(Lock*)

  gibt den Lock frei und legt den Thread schlafen,

  danach Re-Acquire()

  (atomar!)
- Condition::Signal(Lock*)

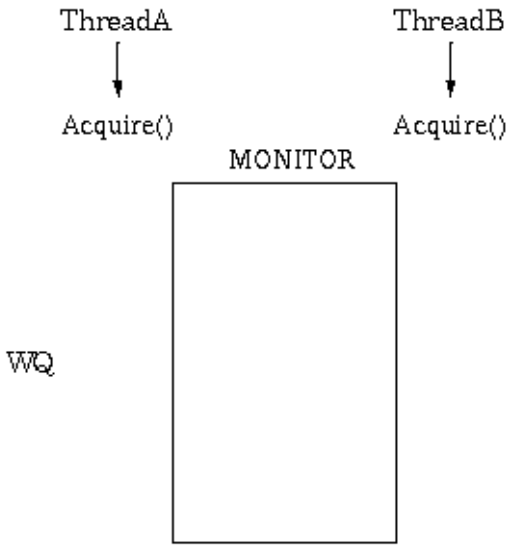
  ersten der wartenden Threads aufwecken (atomar!)
- Condition::Broadcast(Lock*)

  alle wartenden Threads aufwecken (atomar!)

# Task 1 – Conditions (3)

*Ablauf 1*

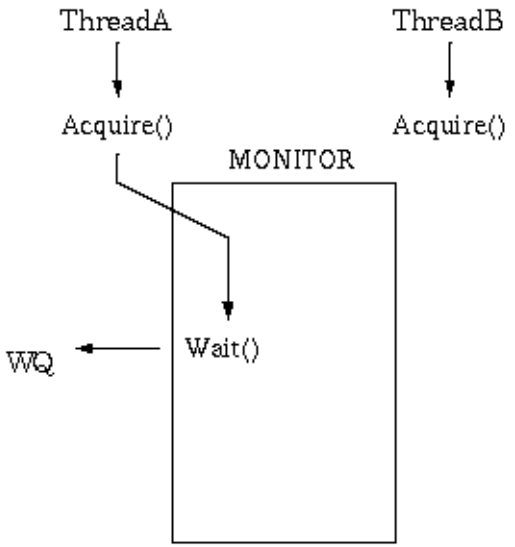
- A & B machen Acquire()
- A erhält den Lock
- B wird demzufolge in die Queue des Lock gelegt



# Task 1 – Conditions (4)

*Ablauf 2*

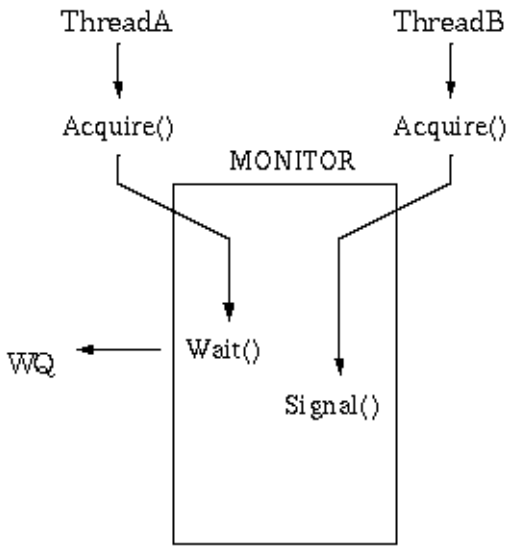
- A macht Wait()
- dabei wird A in die Waiting-Queue der CV gelegt
- vorher wird jedoch der Lock freigegeben, sodass B weiterlaufen kann



# Task 1 – Conditions (5)

*Ablauf 3*

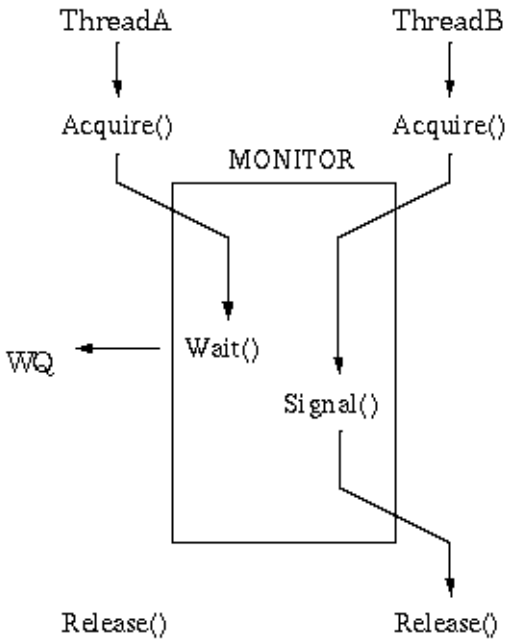
- B läuft weiter und macht irgendwann ein Signal()
- dadurch wird A aus der Waiting-Queue geholt
- da Signal() atomar ist kommt A aber noch nicht zur Ausführung
- ausserdem fordert A sofort nachher – noch in der Wait() - der Lock wieder an und wird wieder blockiert



# Task 1 – Conditions (6)

*Ablauf 4*

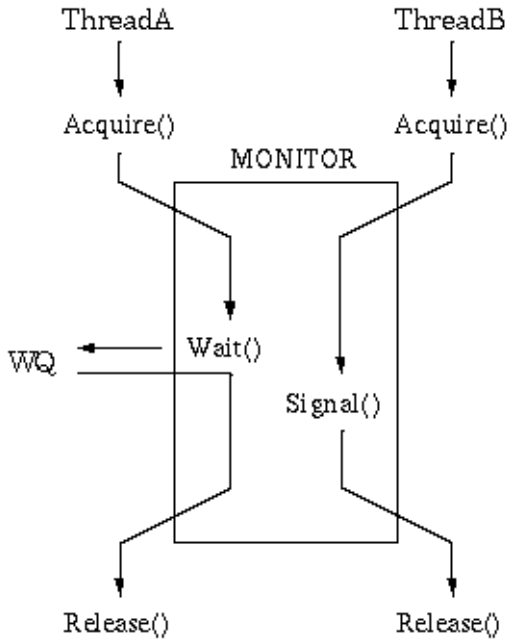
- B läuft weiter und lösten den Lock und die Atomität
- dadurch kann A weiterlaufen



# Task 1 – Conditions (7)

*Ablauf 5*

- A läuft weiter und verlässt die CV
- Schließlich muss A noch den Lock freigeben



# Task 3 – Join

*Wozu Thread::Join()?*

- Möglichkeit, einen Thread auf einen anderen (oder mehrere andere) warten zu lassen
- Synchronisation, z.B. wenn der Thread auf ein Ergebnis des anderen angewiesen ist, um weiterarbeiten zu können
- Join() auf einen bereits beendeten Thread sowie mehrfaches Join() (identisch!) darf keine Auswirkungen haben