Betriebssysteme 1

Foliensatz E, Kap. 5

Prof. Dr. Hans-Georg Eßer

Sommersemester 2021

v3.0 – 27.05.2021

Locking mit TSL / xchg

Listing 5.1: TSL-Implementierung mit xchg

```
1 section .text
2 enter_region:  mov  al, 1
3               xchg al, [lock]
4               cmp  al, 0
5               jne  enter_region
6               ret
7
8 leave_region:  mov  [lock], 0
9               ret
10
11 section .data
12 lock:        db   0
```

enter_region:

- Zeile 2 lädt den Wert 1 in das 8-Bit-Register AL.
- Zeile 3 führt die atomare Tauschoperation aus. Es landet an der Speicheradresse lock also in jedem Fall der Wert 1, und AL enthält danach das vorher in lock gespeicherte Byte (das entweder 0 oder 1 ist).
- In Zeile 4 wird getestet, ob AL den Wert 0 hat.
- Falls nicht, springt die Funktion enter_region zurück zum Anfang und wiederholt die vorherigen Befehle.
- Falls doch, hat die Funktion das Lock erfolgreich „erworben“ und beendet sich.

leave_region ist einfacher gestrickt:

- Zeile 8 schreibt den Wert 0 an die Speicheradresse lock.
- Danach beendet sich die Funktion (Zeile 9).

19. Rekursiver Mutex

- Wenn ein Thread `rec_lock(mutex)` erstmals aufruft, erhält er das Lock und kann weiter arbeiten. Ein interner Zähler im Mutex wird auf 1 gesetzt (für: einmal gesperrt).
- Wenn ein Thread, der mutex bereits erfolgreich gelockt hat, erneut `rec_lock(mutex)` aufruft, wird der interne Zähler erhöht.

23. Java: synchronized

```
1 public class SynchronizedCounter {  
2     private int c = 0;  
3     public synchronized void increment() { c++; }  
4     public synchronized void decrement() { c--; }  
5     public synchronized int value() { return c; }  
6 }
```

Für jedes Objekt ist damit die Member-Variable `c` gegen gleichzeitigen Zugriff durch `increment()`, `decrement()` oder `value()` (aus mehreren Threads heraus) geschützt. Welchen Nachteil hat dieser Ansatz im Vergleich zum Einsatz von Mutexen?