Musterlösungen zu Übung 10 06.07.2011, Hans-Georg Eßer

Übung 10

21 a)

Der Kernel braucht atomare Operationen wie `atomic_inc`, `atomic_set` etc., um den Zugriff auf Kernel-Datenstrukturen zu synchronisieren. Denn auch im Kernel kann es passieren, dass zwischen verschiedenen Teilen des Kernels hin und her gewechselt wird, die mit denselben Daten operieren, und auf Multi-Core- (oder Multi-CPU-) Systemen können auch echt parallel verschiedene Teile des Kernels arbeiten.

Bibliotheken, die Mutexe, Semaphore u. ä. bereitstellen, verwenden die Synchronisationsmethoden, welche der Kernel bereitstellt.

User-Level-Mutexe kann der Kernel nicht verwenden, denn diese sind über eine User-Level-Bibliothek implementiert, die nur zu Prozessen/Threads hinzugeladen werden kann. Funktionen im Kernel können solche Bibliotheken also nicht verwenden. Hinzu kommt noch, dass die Bibliotheken ihrerseits letzten Endes auf Kernel-Funktionen (wie z. B. `test_and_set_ bit`) zurückgreifen, um Mutexe, Semaphore u. ä. zu implementieren (s. o.).

21 b)

`test_and_set_bit` kann verwendet werden, um einen Mutex im Kernel zu implementieren, wobei die beteiligten Kernel-Komponenten hier aktiv warten müssen, wenn das Bit nicht erfolgreich gesetzt werden konnte. Code im Kernel könnte also wie folgt aussehen:

```
bitvector v;
int i = 0;                              // Position 0 in Bitvektor als Mutex

...

while ( ! test_and_set_bit (i, &v) ) {  // versuche, kritischen Bereich zu betreten
  // aktives Warten
}

// kritischer Bereich

clear_bit (i, &v);                      // Zugriff wieder freigeben
```

21 c)

Zum Beispiel kann ein Interrupt-Handler keine klassischen „schlafenden“ Mutexe verwenden, denn er ist kein Prozess oder Thread und könnte vom BS nicht mehr aktiviert werden, nachdem er sich einmal schlafen gelegt hätte. Die Komponenten des BS, die nicht schlafen können, dürfen darum nur aktiv warten.

21 d)

Reader Writer Locks erlauben gegenüber normalen Locks feiner granulares Sperren von Ressourcen. Damit lässt sich z. B. regeln, dass mehrfach lesend auf ein und dieselbe Ressource zugegriffen wird, während ein Schreibzugriff nur erlaubt ist, wenn es keine anderen Zugriffe gibt.

21 e)

Kernel-Semaphore sind „schlafende“ Locks, sie dürfen also nur in Kernel-Komponenten verwendet werden, die schlafen können. Wie im User-Space wird auch hier für jeden Semaphor eine Warteschlange verwaltet.
`down_trylock()` arbeitet wie `down()`, kehrt aber in jedem Fall sofort zurück; die aufrufende Funktion muss prüfen, ob das Lock erworben wurde oder nicht.