Alexander Züpke

Abstract: Futexe für partitionierte, sichere Betriebssysteme

Dieser Vortrag beschreibt, wie das aus Linux bekannte Fast User Space Mutex (Futex)-Konzept für partitionierte Betriebssysteme und Mikrokerne umgesetzt werden kann. Die vorgestellte Lösung zeigt ein Zeitverhalten von O(1) bei linearem Speicherverbrauch.

Der Fast User Space Mutex (Futex)-Mechanismus in Linux ermöglicht leichtgewichtige Synchronisationsobjekte wie Mutexe für nebenläufige Anwendungen (Multithreading). Solange nur ein Thread auf ein Synchronisationsobjekt zugreift, kann der Zustand des Synchronisationsobjektes durch atomare Operationen manipuliert werden, ohne dass ein teurer Aufruf des Betriebssystemkerns notwendig wäre. Erst wenn weitere Threads einen schon gesperrten kritischen Bereich betreten wollen, müssen diese Threads den Betriebssystemkern aufrufen. Der Kern legt dann bei Bedarf ein Warteschlangenobjekt an und suspendiert den Aufrufer. Genauso muss beim Verlassen eines kritischen Bereiches geprüft werden, ob Threads warten, und der Betriebssystemkern aufgerufen werden, um den nächsten suspendierten Thread zu wecken. Allerdings ist die Implementierung in Linux nicht für partitionierte Betriebssysteme geeignet. Solche partitionierten Betriebssysteme (z.b. ARINC 653) werden in sicherheitskritischen Anwendungen im Bereich der Luft- und Raumfahrt eingesetzt und zeichnen sich durch ein striktes Prozessmodell aus, bei dem alle Betriebsmittel vorab statisch auf sogenannte Partitionen verteilt werden. Dynamisch erzeugte Warteschlangenobjekte stellen hier ein Hindernis bei der Partitionierung dar.

Der Vortrag zeigt, wie sich die Futex-Implementierung von Linux so abwandeln lässt, dass kein Speicherallokator für Warteschlangenobjekte mehr benötigt wird, aber trotzdem eine sichere Partitionierung möglich ist. Der vorgestellte Ansatz ist dann auch für Mikrokerne oder eingebettete Systeme nutzbar. Basierend auf den Futexen wird eine Implementierung von blockierenden Mutexen und Condition-Variablen mit linearem Speicherverbrauch pro wartendem Thread und O(1) Zeitverhalten vorgestellt.