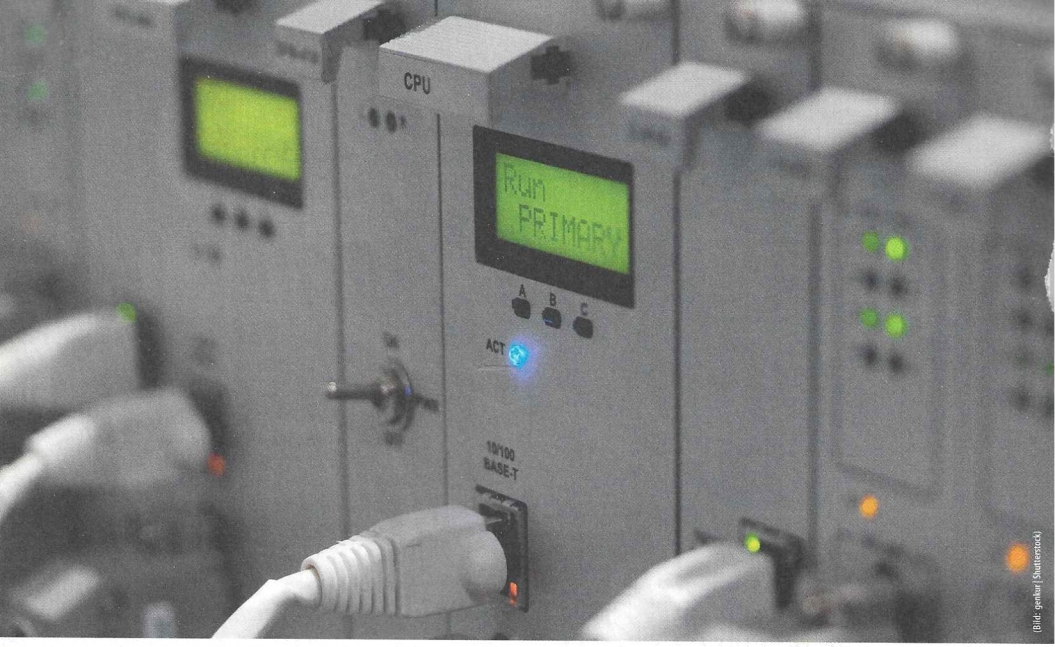

(Bild: gerkan | Shutterstock)

Bewährte Werkzeuge bei Embedded-Systemen:

Linux-Betriebssysteme sicher updaten

Embedded-Systeme laufen oft mit dem Linux-Betriebssystem. Um die volle Sicherheit zu gewährleisten, benötigt das System zuweilen Updates. Kritisch wird es, wenn während des Update-Vorgangs etwas schief geht. Aber das lässt sich mit bewährten Mitteln vermeiden.

Von Jan Altenberg

Der Einzug moderner und leistungsfähiger SoCs in Embedded-Systeme hat die Neuentwicklung und die Pflege von Produkten nachhaltig verändert. Der Funktionsumfang moderner Prozessoren läßt sich nur mit einem Betriebssystem effizient nutzen. Linux ist für diesen Einsatzzweck prädestiniert. Kein anderes Betriebssystem bietet eine so hohe Portierbarkeit und eine vergleichbare Anzahl an verfügbaren Treibern. Noch nie war es einfacher, die Anbindung an ein Netzwerk oder die Integration einer grafischen Oberfläche umzusetzen. Doch dieser Umstand bringt auch einige Nachteile mit sich: Neue Schnittstellen machen ein System auch angreifbarer und die Komplexität eines Betriebssystems und der dazugehörigen Programme erhöhen auch die Fehleranfälligkeit.

Nur noch ein Bruchteil der im Produkt zum Einsatz kommenden Software-Komponenten wird noch selbst erstellt, die meisten Programme werden nur noch angepasst und integriert. Auch Linux-Systeme bestehen zu einem großen Teil aus solchen Komponenten. Und deren Komplexität birgt natürlich auch das Risiko potenzieller Sicherheitslücken. Ein System ohne die Möglichkeit Software-Aktualisierungen ins Feld zu bringen, muss unter diesen Gesichtspunkten als grob fahrlässig betrachtet werden. Ein Sicherheitskonzept besteht daher also nicht nur aus der einmaligen Absicherung bzw. Abhärtung eines Systems, sondern insbesondere auch aus der Möglichkeit, fehlerhafte oder risikobehaftete Komponenten austauschen zu können.

Umsetzung von Updates ohne Bootloader

Die wichtigste Anforderung an ein Update-Konzept ist die Robustheit. In einer Fehlersituation muss mindestens die Möglichkeit bestehen, ein Update zu wiederholen. Auf gar keinen Fall darf ein fehlgeschlagenes Update zum Ausfall eines Gerätes führen. Traditionell wurden Updates daher über den Bootloader eingespielt. Dieser wurde als „atomare" und nicht austauschbare Komponente betrachtet und bildete damit den Fallback bei einem fehlge-

schlagenen Update. Der Bootloader war immer vorhanden und somit bestand immer die Möglichkeit ein Update einzuspielen. Technisch ist dies durchaus eine valide Lösung, bringt aber einige Nachteile mit sich: Die Umsetzung ist in der Regel sehr gerätespezifisch und für neue Geräte- oder Produktgenerationen entsteht ein hoher Portierungsaufwand. Weiterhin müssen alle Treiber mehrfach implementiert werden. Einmal im Betriebssystem und einmal im Bootloader; zumindest für jegliche Peripherie, die beim Einspielen von neuer Software beteiligt ist.

Nun ist die Umsetzung eines Update-Konzeptes aus dem Bootloader technisch zwar eine akzeptable Lösung, es besteht aber keine Notwendigkeit dafür. Dieses Vorgehen hat sich historisch so entwickelt, da Speicherplatz oft limitiert und sehr teuer war. In Anbetracht der Tatsache, dass moderne Systeme mit deutlich mehr Speicher ausgestattet sind, bestehen ganz neue Möglichkeiten. Grundsätzlich geht man bei Linux-basierten Systemen daher dazu über, ein Update nicht mehr aus dem Bootloader, sondern aus einem redundanten Linux-System einzuspielen, welches durchaus sehr minimalistisch gestaltet sein kann.

Somit können alle Treiber wiederverwendet werden und es stehen für die Umsetzung alle Möglichkeiten zur Verfügung, die Linux bietet. Die stabilen Schnittstellen zur Applikation sorgen dafür, dass ein solches Konzept auch für neue Gerätegenerationen wiederverwendet werden kann, da die Implementierung weitestgehend generisch erfolgt. Konzeptionell lassen sich daraus zwei Lösungswege ableiten: Entweder Implementierung eines redundanten Systems, oder Implementierung eines Rescue-Systems.

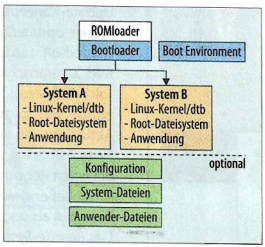

Bild 1. Aufbau eines symmetrischen Updates (redundantes System). (Quelle: Linutronix)

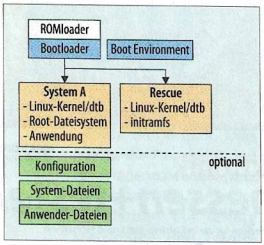

Bild 2. Aufbau eines asymmetrischen Updates (Rescue-System). (Quelle: Linutronix)

Update mit redundantem System

Aus Sicht des Anwenders ist ein redundantes System die angenehmste Lösung für die Umsetzung eines Updates. Bei dieser Variante bleibt das Produkt auch bei einer fehlgeschlagenen Aktualisierung immer funktional. Bei einem redundanten System, häufig auch als symmetrisches Update oder A-B-Update bezeichnet, werden zwei voll funktionale Software-Stände parallel gehalten. Der aktive Software-Stand aktualisiert den jeweils redundanten Pfad.

Bild 1 zeigt den Aufbau eines symmetrischen Updates. Das Einspielen der Software passiert analog zum Rescue-System. System A kann die Software von einem definierten Ort laden. System B wird dann mit den neuen Daten überschrieben und zeitgleich als ungültig markiert. Anschließend wird im Boot Environment festgelegt, dass System B gebootet werden soll. Der anschließende Reboot wird wiederum per Watchdog abgesichert. Im Erfolgsfall markiert sich System B als gültig, im Fehlerfall erkennt der Bootloader den Watchdogreset. Anhand der Tatsache, dass System B noch nicht gültig ist, kann der Bootloader entscheiden, wieder System A zu booten. Nachteilig ist bei dieser Methode nur der erhöhte Speicherbedarf, da die Software doppelt vorgehalten werden muss. Positiv ist, dass pro Update nur ein Reboot benötigt wird und das System auch im Fehlerfall immer voll funktionsfähig ist.

Update mit Rescue-System

Für alle Systeme mit begrenztem Speicher ist die Umsetzung eines asymmetrischen Updates die richtige Wahl. Bei einem sogenannten „Rescue-System" wird parallel zum Produktivsystem ein minimalistisches, redundantes Linux abgelegt. Dies besteht aus einem einfach konfigurierten Linux-Kern und einem Ramfilesystem. Das Ramfilesystem besitzt alle notwendigen Werkzeuge zur Durchführung eines Updates. Das Einspielen von neuer Software erfolgt grundsätzlich aus dem Rescue-System. Dieser Weg erfüllt die Minimalanforderung an ein

Update-Konzept. Das Gerät bleibt über das Rescue-System immer updatefähig. Schlägt das Update fehl, kann dieses wiederholt werden.

Bild 2 veranschaulicht den grundsätzlichen Aufbau eines solchen Systems. Die Aktualisierung der Software in System A würde wie folgt ablaufen: System A lädt die Software-Aktualisierung von einem definierten Ort. Dabei legt das System über ein Setting im Boot Environment fest, dass beim nächsten Reboot das Rescue-System gebootet werden soll. Anschließend wird, ebenfalls im Bootloader Environment, System A als ungültig markiert und ein Reboot angestoßen. Das Rescue-System spielt das Update ein und überschreibt damit die Daten von System A. Die „Bootweiche" wird auf System A zurückgesetzt und ein Reboot wird angestoßen. Dieser Reboot wird über einen Watchdog abgesichert. Im Erfolgsfall wird System A am Ende des Bootprozesses sich selbst als gültig markieren. Im Fehlerfall wird der Watchdog das System resetten. Ein Fehlerfall ist beispielsweise das Einspielen einer fehlerhaften Software. Stellt der Bootloader als Reset-Grund einen Watchdogreset fest und ist System A noch nicht als gültig markiert, so wird von einem fehlerhaften Update ausgegangen und das Rescue-System wird gebootet. Dieses kann dazu genutzt werden, wieder eine korrekte Software einzuspielen.

Bei einer solchen Umsetzung gibt es zwei Vorteile. Der Erste ist ein geringer Speicherbedarf. Der Zweite ist die Möglichkeit, Produktivsystem und Rescue-System unterschiedlich zu konfigurieren und mit unterschiedlichen Zugriffsrechten zu versehen. Nachteilig ist zu erwähnen, dass das Produktivsystem während des Updates und im Fehlerfall nicht nutzbar ist. Außerdem sind für jedes Update mindestens zwei Reboots erforderlich. Für die meisten Anwendungsfälle sind diese Einschränkungen aber durchaus akzeptabel.

Verwenden von bewährten Werkzeugen

Bedingt durch die Tatsache, dass sich Updates aus Linux heraus sehr generisch gestalten lassen, steht bereits eine Vielzahl von Werkzeugen zur Verfügung. Die Werkzeuge können bei der Umsetzung eines Update-Konzeptes helfen. Die Verwendung solcher Tools spart manuelle Aufwände und reduziert die Fehleranfälligkeit. Ein sehr verbreitetes und bewährtes Tool bzw. Framework zur Umsetzung von Updates ist „swupdate", welches unter einer Open-Source Lizenz frei verfügbar ist (https://github.com/sbabic/swupdate). Swupdate unterstützt das Einlesen und Einspielen von Updates über ein einheitliches und streamfähiges Containerformat. Außerdem können Software-Stände signiert werden, um das Einspielen von Schadsoftware zu verhindern. Swupdate greift dabei durchweg auf frei verfügbare Technologien und bewährte Standards zurück.

Verteilung von Updates

Bevor ein Update eingespielt werden kann, muss es zuerst einmal auf das Gerät kommen. Das zuvor erwähnte Swupdate-Tool unterstützt den Anwender nicht nur beim Einspielen von Updates, sondern auch beim Deployment. Der einfachste Fall ist das Verarbeiten einer lokalen Datei, beispielsweise von einem USB-Stick oder einem ähnlichen

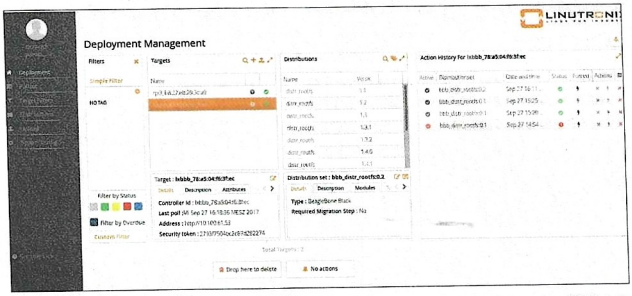

Bild 3. individuell angepasstes Hawkbit User-Interface. (Bild: Linutronix)

Speichermedium. Weiterhin können Updates per Netzwerk über das push-Modell oder das Pull-Modell eingespielt werden. Beim Pull-Modell werden die Daten von einer Server URL abgeholt. Beim Push-Modell geschieht der Upload der Daten via Webserver. Für den Upload der Daten über einen Webserver steht bereits ein einfaches Web-User-Interface zur Verfügung.

Darüberhinaus besteht die Möglichkeit einer Anbindung an den Deployment Server Hawkbit. Hawkbit ist ebenfalls ein Open-Source-Projekt und beschäftigt sich mit dem Roll-out von Software. Über ein serverbasiertes, grafisches Interface können Geräteklassen verwaltet und bestimmte Software-Stände an bestimmte Geräte ausgeliefert werden. Swupdate unterstützt die Anbindung an Hawkbit out-of-the-box. Das grafische Interface von Hawkbit lässt sich individuell anpassen. **Bild 3** zeigt ein solches angepasstes Hawkbit-User-Interface.

Wann ist ein Update nötig?

Die in den bisherigen Abschnitten beschriebenen Methoden zum Einspielen neuer Software-Stände sind allerdings nur eine Seite der Medaille. Bevor überhaupt ein Update eingespielt werden kann, ist noch ein ganz anderer Punkt zu beachten: Wie und auf welcher Grundlage wird denn überhaupt entschieden, ob ein Update einer Drittkomponente, wie zum Beispiel der C-Bibliothek, des Webservers oder der verwendeten Grafikbibliothek, notwendig ist? Der Aufwand für das hierfür notwendige Tracking von Sicherheits-Patches darf nicht unterschätzt werden. Es empfiehlt sich daher grundsätzlich auf gewartete Software-Pakete zurückzugreifen.

Bewährt haben sich die Pakete der Debiandistribution. Sicherheitskritische Updates sind bei Debian sehr schnell verfügbar und Updates lassen sich sehr einfach und automatisiert tracken. Für den Linux-Kern ist die Verwendung einer sogenannten Longterm-stable-Version die richtige Wahl. Diese werden mindestens zwei Jahre gepflegt. Welche Kernelversionen aktuell über eine Langzeitpflege verfügen, läßt sich unter http://kernel.org einsehen. Selbstverständlich muss nicht jedes dieser Sicherheits-Updates integriert werden. Die oben beschriebenen Aktualisierungen bieten aber eine solide Entscheidungsgrundlage.

Bewährte Werkzeuge machen Updates sicherer

Sicherheit und Pflege sind wesentliche Aspekte des Produktdesigns. Updates sind daher ein zwingender Bestandteil von Embedded-Systemen. Eine wichtige Grundlage für die Bereitstellung von Updates ist die Verwendung von gewarteten Software-Paketen, zum Beispiel die Pakete aus der Debian Distribution oder die Longterm-stable-Versionen des Linux-Kerns. Updates werden üblicherweise nicht mehr aus dem Bootloader, sondern aus einem redundanten Linux-System eingespielt. Dies erhöht die Flexibilität und die Wiederverwendbarkeit. Zudem kann bei der Implementierung auf fertige Werkzeuge und Frameworks, wie z.B. Swupdate, zurückgegriffen werden. Diese Werkzeuge unterstützen bereits alle gängigen Speichermedien und Deployment-Methoden. Die Verwendung bewährter Konzepte und bewährter Tools reduziert die Implementierungsaufwände drastisch und verringert zeitgleich die Fehleranfälligkeit. *jk*

Jan Altenberg
beschäftigt sich seit mehr als 15 Jahren beruflich mit Linux. Er betreute u.a. verschiedene Arbeiten für das EU-Projekt OCEAN, welches sich zum Ziel setzte, eine offene Steuerungsplattform auf Basis von Realtime Linux zu schaffen. Weiterhin trug er bei einem namhaften Maschinenhersteller die Verantwortung für die Einführung von Linux in der neuen Steuerungsgeneration. Seit 2007 leitet er bei Linutronix den technischen Vertrieb. Altenberg ist regelmäßiger Sprecher auf verschiedenen Fachkonferenzen und wurde 2014 von den Besuchern des ESE Kongress mit dem Speaker Award Publikumspreis ausgezeichnet.