Konzepte von Betriebssystem-Komponenten

QNX

Martin Mitzlaff
martin.mitzlaff@informatik.stud.uni-erlangen.de

28.11.2002

1. Was ist QNX?

QNX ist ein „Microkernel-Echtzeit-Unix“, das von der kanadische Firma QNX Software Systems Ltd. entwickelt wird. Gegründet wurde die Firme 1980 von Gordon Bell und Dan Dodge, die damals das Betriebsystem QUNIX entwickelt hatten. Heute verkauft QNX sein Betriebsystem in mehr als 100 Staaten.
QNX ist aber eigentlich kein UNIX-Betriebssystem, sondern „nur“ POSIX kompatibel. POSIX legt nur eine Schnittstelle fest und kein Implementierung. QNX ist also kein UNIX-Derivat. Durch die POSIX-Kompatiblität kann aber Source-Code von andren UNIX-Systemen leicht portiert werden und die Umgewöhnung für Entwickler und Nutzer von Unix-Derivaten zu QNX ist nicht sehr kompliziert.
QNX basiert auf einem Microkernel, wodurch es sehr skalierbar und flexibel ist. Sowohl minimal Konfigurationen (Kernel + wenige Module) für eingebettete Systeme, als auch voll ausgebaute netzwerkweite Systeme mit mehr als 100 Usern sind möglich.
Einsatz findet QNX hauptsächlich als embedded System in der Kommunikationstechnik, Luftfahrt, Medizintechnik und Consumer Electronic. Firmen, die Produkte mit QNX anbieten, sind z.B. DASA, HP, Motorola und Siemens.
Neu ist der Einsatz in der Automobiltechnik. Wo es hauptsächlich in Navigations-, Unterhaltungs- und Telematiksystemen zum Einsatz kommt.
In der Forschung basieren diverse autonome Roboter auf QNX.
Inzwischen hat die jüngste Version von QNX-RTOS (Real-Time OS) auch Einzug auf Hunderttausenden von Desktops gehalten: Seit Ende September 2000 bietet QNX sein Betriebssystem für Entwickler und Privatanwender zum kostenlosen Download (http://www.qnx.com/nc) an. Dieser Schachzug soll dem Betriebssystem zu größerer Verbreitung und einer umfassenderen Software-Unterstützung verhelfen. Schon in den ersten drei Tagen wurden über 200.000 Kopien vom Server gezogen.

2. Was ist ein Echtzeit-Betriebsystem?

Ein Echtzeit-Betriebssystem muss innerhalb einer vorhersagbaren, definierten Zeitspanne (maximale Latenzzeit) auf eine Unterbrechungsanforderung reagieren. Das heißt, dass der Rechner zu einem vorgegebenen Zeitpunkt spätestens seine Arbeit verrichtet haben muss. Was aber nicht bedeutet, dass es ein besonders schnelles System sein muss, es muss nur bestimmte Zeitvorgaben einhalten.
Beim normalen Umgang mit PCs spielt dessen Timing-Verhalten keine große Rolle. Anders sieht das z.B. bei Steuerungscomputern in der Medizin oder Industrie aus. Wenn hier ein für das System kritischer oder relevanter Zustand eintritt, muss es innerhalb einer gewissen Zeitspanne – je nach Komplexität und Wichtigkeit in der Zeitspanne von Mikro- bis Millisekunden – darauf reagieren. Dabei muss sich die Echtzeit-Anwendung natürlich auf das Betriebssystem verlassen können.

3. Der QNX-Kernel und sein Umfeld

QNX besteht aus einem kleinen Kern mit einer Gruppe kooperierender Prozesse. Eine typische QNX Konfiguration beinhaltet die folgenden Systemprozesse: Prozess-Manager, Dateisystem-Manager, Geräte-Manager und den Netzwerk-Manager.
Gerätetreiber sind Prozesse, welche das Betriebssystem von dem Umgang mit den Details für die Unterstützung spezifischer Hardware abschirmen. Da Treiber wie normale Prozesse starten, hat das Hinzufügen neuer Treiber zu QNX keinen Einfluss auf andere Teile des Betriebssystems. Alle Module (außer dem Prozess-Manager) können während der Laufzeit gestartet und beendet werden.

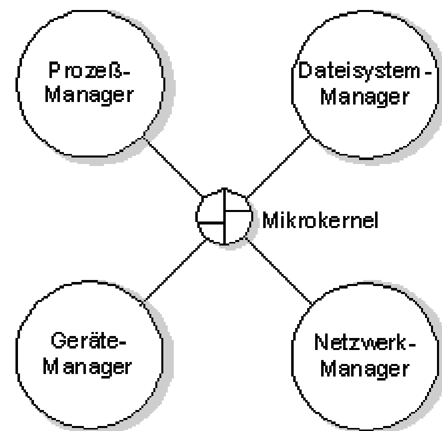

Abb 1: Der QNX Microkernel koordiniert die System-Manager.

Systemprozesse unterscheiden sich praktisch nicht von selbstgeschriebenen Anwendungsprogrammen der Benutzer - sie haben keine privaten oder versteckten Schnittstellen, welche für Benutzerprozesse nicht verfügbar wären. Genau diese Architektur ist es, die QNX seine Skalierbarkeit verleiht. Weil die meisten Betriebssystemdienste durch Standard- QNX-Prozesse angeboten werden, ist es sehr leicht, das Betriebssystem zu erweitern: Um neue Dienste anzubieten, schreibt man einfach ein eigenes Programm. Tatsächlich kann die Grenze zwischen dem Betriebssystem und den Applikationen leicht ineinander übergehen.

3.1. Microkernel

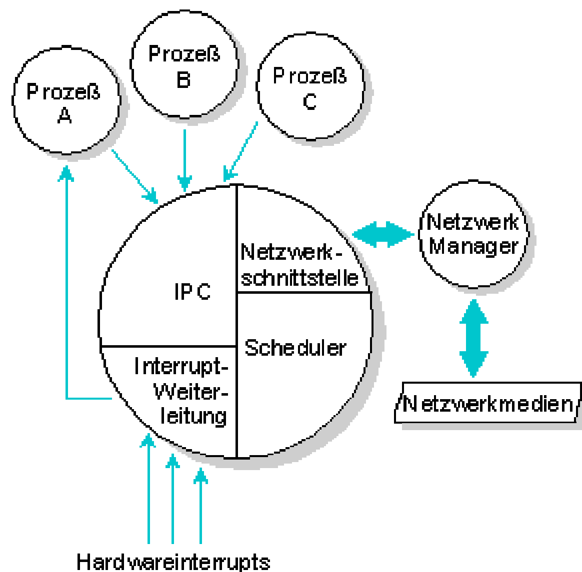

Abb 2: Das Innenleben des QNX Microkernel

Der QNX Microkernel übernimmt folgende Aufgaben:

- IPC - der Microkernel überwacht das Routing von Nachrichten; er verwaltet Proxies und Signale
- Netzwerkkommunikation auf unterster Stufe - der Microkernel liefert alle Nachrichten für Prozesse auf anderen Knoten aus
- Prozess-Scheduling
- Handhabung von Interrupts auf erster Stufe - alle Hardwareinterrupts und Schutzverletzungen werden zuerst durch den Microkernel geleitet und dann an den entsprechenden Treiber oder System-Manager weitergeleitet

3.2. IPC Inter-Prozess-Kommunikation

QNX unterstützt einen einfachen aber leistungsfähigen Satz von IPC-Werkzeugen, welche die Entwicklung von Applikationen, die aus kooperierenden Prozessen bestehen, sehr einfach gestalten.
QNX war das erste kommerzielle Betriebssystem seiner Art, welches Message Passing als fundamentale Grundlage für die IPC benutzt. QNX verdankt vieles seiner Leistung, Einfachheit und Eleganz der kompletten Integration des Message Passing innerhalb des gesamten Systems.

3.2.1. IPC über Nachrichten

Eine Nachricht besteht aus einem oder mehreren Bytes, die von einem Prozess zu einem anderen übertragen werden. Die Bedeutung der übertragenen Bytes kennen nur die beiden beteiligten Prozesse, das Betriebssystem sorgt nur für den Transport. Zur Kommunikation kooperierender Prozesse stehen folgende C-Funktionen zur Verfügung, die sowohl lokal als auch über das Netzwerk genutzt werden können:

send()	Zum Senden von Nachrichten
receive()	Zum Empfang von Nachrichten
reply()	Zum Beantworten einer empfangenen Nachricht

Prozess A will an Prozess B eine Nachricht schicken:
Prozess A ruft *send()* auf. Darin teilt er mit an wen er eine Nachricht schicken will (PID von Prozess B), den Nachrichten- und den Empfangspuffer. Prozess A ist jetzt im Zustand SEND-blockiert. Wenn Prozess B bereit ist Nachrichten zu empfangen, führt er *receive()* aus. Dies enthält, von wem er Nachrichten empfängt und in welchem Puffer er diese entgegennimmt. Prozess B empfängt jetzt die Nachricht und Prozess A wechselt in den Zustand REPLY-blockiert. Prozess B verarbeitet die Nachricht, und initiiert ein *reply()*. Die Antwortnachricht wird zu Prozess A kopiert, welcher umgehend in den Zustand READY versetzt wird. Die Funktion *reply()* blockiert nicht, so dass auch Prozess B in READY geht.

3.2.2. IPC über Proxies

Mit Proxies kann ein Nachrichtenaustausch zwischen Prozessen stattfinden, ohne den sendenden Prozess zu blockieren. Der Proxy sendet eine feste Nachricht, die man bei seiner Erzeugung übergeben hat, an den Prozess, der das Proxy besitzt. Ein Proxy kann mehrmals hintereinander mit der Funktion *trigger()* dazu veranlasst werden seine feste Nachricht zu senden. Dabei können bis zu 65.535 Nachrichten in der Warteschlange stehen.

3.2.3. IPC über Signale

IPC mit Signalen ist die traditionelle Form der Kommunikation, die schon in einer Vielzahl von Betriebssystemen eingesetzt wird. QNX unterstützt POSIX-kompatible und proprietäre Signale.
Ein Prozess erhält das an ihn gesendete Signal, wenn er durch den Scheduler zur Ausführung kommt. Über die Reihenfolge des Versendens anstehender Signale kann keine Aussage gemacht werden.
Mit dem Signal SIGSEGV (Entdeckung einer ungültigen Speicherreferenz) hat wahrscheinlich schon jeder unangenehme Bekanntschaft gemacht. Ein Prozess erhält vom Betriebssystem dieses Signal, wenn ein Prozess in unzulässiger Weise auf einen Speicherbereich zugreifen möchte. Wird diese Situation nicht richtig behandelt, wird der Prozess beendet.

3.3. Scheduling

In QNX wird jedem Prozess eine Priorität zugeordnet. Der Scheduler entscheidet, welcher Prozess als nächster läuft, indem er sich die Priorität aller Prozesse, welche READY sind, ansieht. Der Prozess mit der höchsten Priorität wird gewählt. Die Standardpriorität eines neuen Prozesses wird ihm von seinem Vater Prozess vererbt. Die Priorität kann der Prozess selbst ändern. So kann ein Server-Prozess z.B. seine Priorität ändern, wenn eine Nachricht (Anfrage) erhält.
Wenn verschiedene Prozesse die gleich Priorität haben, können diese nach drei verschiedenen Schedulingverfahren ausgewählt werden:
FIFO-Scheduling: Der ausgewählte Prozess läuft solang, bis er die Ausfühungskontrolle freiwillig abgibt oder durch einen Prozess mit höherer Priorität verdrängt wird. Dadurch wird der gegenseitige Ausschluss bei Zugriff auf Ressource ohne Semaphoren zur Hilfe zu nehmen garantiert.
Round-Robin-Scheduling: Der ausgewählt Prozess wird so lang ausgeführt bis, bis er die Ausfühungskontrolle freiwillig abgibt, durch einen Prozess mit höherer Priorität verdrängt wird oder seine Zeitscheibe abläuft. Wenn der Prozess seine Zeitscheibe aufgebraucht hat, wird er verdrängt und der nächste laufwillige Prozess mit dem Zustand READY und der gleichen Priorität wird zur Ausführung eingeplant. Eine Zeitscheibe beträgt 50 Millisekunden.
Adaptives Scheduling: Wenn ein Prozess seine Zeitscheibe aufgebraucht hat (und er bis dahin nicht blockiert), wird seine Priorität um 1 verringert. Wichtig hierbei ist, dass die Verringerung nur um eine Prioritätsstufe erfolgt, auch wenn der Prozess eine weitere Zeitscheibe verbraucht - die Priorität wird maximal um eins niedriger gesetzt, als die Ausgangspriorität. Blockiert ein Prozess, bekommt er umgehend seine Ausgangspriorität zurück.

3.4. Timing beim Auftreten von Interrupts

Computer können leider noch nicht in unendlich kleiner Zeit auf Ereignisse reagieren, so dass auch in einem richtigen Echtzeit-Betriebssystem eben eine gewisse Zeit vergeht, bis ein Programm auf ein externes Ereignis reagieren kann. Bei QNX hat der Hersteller aber darauf geachtet, diese Latenzzeit möglichst kurz zu halten.
Die Interrupt-Latenzzeit wird als die Zeitdifferenz zwischen dem Eintreffen (und Akzeptieren) eines Hardwareinterruptes und dem Ausführen der ersten Instruktion eines Software-Interrupthandlers definiert. QNX beläßt alle Interrupts aktiviert, so dass die Interruptlatenzzeit keine signifikante zeitliche Auswirkung hat. Es gibt jedoch manchmal die Forderung, dass Codeabschnitte gegen das Eintreffen von weiteren Interrupts geschützt werden müssen. Hierfür wird kurzzeitig ein Interrupt gesperrt, um die kritische Codesequenz auszuführen. Diese Sperrzeit definiert den schlechtesten Fall einer Interrupt-Latenzzeit - in QNX ist diese sehr klein.
In manchen Fällen ist es erforderlich, dass ein Interrupthandler einen regulären Prozess benachrichtigen, bzw. anwerfen muss. Dies geschieht, indem der Interrupthandler einen Proxy des wartenden Prozess triggert. Dies führt zu einer weiteren Verzögerung – der Scheduling-Latenzzeit.
Die Scheduling-Latenzzeit ist die Zeit zwischen der Beendigung eines Interrupthandlers und der Ausführung des ersten Befehls eines Treiberprozesses. Diese Zeitdifferenz entsteht, da der Prozesskontext des augenblicklich laufenden Prozesses gesichert und der neue Kontext für den zu startenden Prozess wiederhergestellt werden muss.

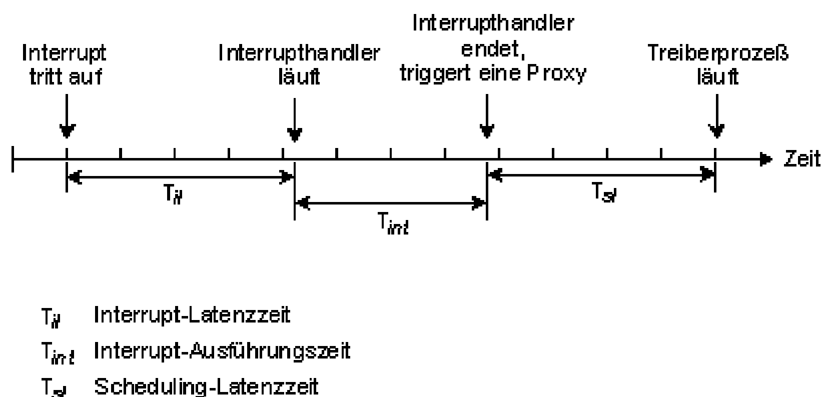

Abb 3: Interrupthändler gibt einen Proy zurück, wenn er terminiert.

Da Mikrocomputerarchitekturen es zulassen, für Hardwareinterrupts Prioritäten zu vergeben, können hochpriorisierte Interrupts niedriger priorisierte verdrängen.

3.5. Netzwerk

Der Netzwerk-Manager bietet QNX-Benutzern eine nahtlose Erweiterung des leistungsstarken Nachrichtensystems des Betriebssystems. Da er direkt mit dem Microkernel kommuniziert, erweitert er die IPC-Fähigkeiten von QNX an entfernte Computer.
Der Netzwerk-Manager ist ein unabhängiges Modul, es muss nicht in das Image des Betriebssystems eingebaut werden. Vielmehr kann er jederzeit gestartet und angehalten werden, um die Nachrichtenübermittlung im Netzwerk bereitzustellen oder zu entfernen. Wenn der Netzwerk-Manager startet, registriert er sich bei Prozess-Manager und dem Kernel.

Eine QNX Applikation kann mit einem Prozess auf einer anderen Maschine im Netzwerk genauso kommunizieren, als ob sie mit einem anderen Prozess auf der gleichen Maschine kommuniziert. Tatsächlich gibt es aus der Sicht der Applikation keinen Unterschied zwischen lokalen und entfernten Ressourcen. Dieser bemerkenswerte Grad an Transparenz wird durch virtuelle Verbindungen (VCs) ermöglicht, welche Pfade sind, die der Netzwerk-Manager für die Überbringung von Nachrichten, Proxies und Signalen über das Netzwerk, anbietet.
Die Endpunkte einer VC sind virtuelle Prozesse (VID). Der Nachrichtenaustausch selbst findet mit dem VID statt. Der Netzerk-Manager sorgt dann für die Weiterleitung der Nachrichten an den anderen VID. Dieser sendet die Nachricht dann an den Zielprozess.
Eine virtuelle Proxy erlaubt es, eine Proxy von einem entfernten Knoten aus anzusprechen, fast wie es eine virtuelle Verbindung einem Prozess erlaubt, Nachrichten mit einem entfernten Knoten auszutauschen. Im Gegensatz zu einer virtuellen Verbindung, welche zwei Prozesse miteinander verbindet, erlaubt eine virtuelle Proxy jedem Prozess auf dem entfernten Knoten, diesen zu triggern.
Der Netzwerk-Manager sorgt außerdem für einen Lastenausgleich zwischen verschiedenen Kommunikationswegen, für Fehlertoleranz durch redundante Verbindungen und für Überbrückung zwischen verschiedenen QNX-Netzwerken.
Zur Verbindung mit anderen System gibt es natürlich auch einen TCP/IP-Manager.

4. Literatur:

1. QNX System Architektur: QNX Software Systems GmbH, 1999, http://www.qnx.de/literatur/qnx4_sysarch_de/index.html

2. System Architecture (QNX 6.20): QNX Software Systems Ltd., 2002, http://www.qnx.com/developer/docs/momentics_nc_docs/neutrino/sys_arch/about.html

3. System Architecture (QNX 4.25): QNX Software Systems Ltd., 2000, http://www.qnx.com/developer/docs/qnx_4.25_docs/qnx4/sysarch/about.html

4. Jörg Luther : QNX: Echtzeit-OS zum Nulltarif, März 2002, http://www.tecchannel.de/betriebssysteme/582/index.html

5. QNX: http://www.beringer-engineering.ch/5_products/d_ebene2/qnx.htm

6. Falk Sippach: QNX – Das Echtzeitbetriebssystem, April 2001, http://www.uni-weimar.de/~sippach1/uni/qnx/