Ausgewählte Kapitel der praktischen Betriebssystemprogrammierung (AKBPII)

# PCCard: Architektur und Treiberdesign in JX

**Michael Schmidt**
swmeschm@cip.informatik.uni-erlangen.de

22. März 2004

## 1 Überblick

In Rahmen des Hauptseminares “Ausgewählte Kapitel der praktischen Betriebssystemprogrammierung 2” sollte die PCMCIA-Unterstützung für das Betriebssystem JX implementiert werden - hierbei wurde beispielhaft der Texas Instruments Chip 1250 verwendet.

Die PCMCIA-Sockets in mobilen Computern sind PCI-zu-PCMCIA bzw. PCI-zu-CardBus Brücke und bieten dem Entwickler die standardisierte PCI-Schnittstelle zur Programmierung an.

## 2 PCI und PCMCIA

### 2.1 PCI

Der PCI-Bus (Peripheral Component Interconnect) ist ein weitgehend CPU-unabhängiges hochperformantes Bussystem für Computer und wurde im Jahre 1991 in seiner ursprünglichen Form von Intel konzipiert und im Laufe der Jahre stetig weiterentwickelt.

Für die Implementierung der PCMCIA-Sockets sind vor allem die Zugriffsmöglichkeiten auf vom PCI-Bus verwaltete Geräte relevant. Dies geschieht über den sogenannten “Configuration Space”, welcher für jedes Gerät existiert. Unter Verwendung der PCI-Adresse (3-Tupel mit Busnummer, Gerätenummer und Funktionsnummer) kann man über das Schreiben dieser Adresse in das CONFIG_ADDRESS (0xcf8h) Register entsprechend die Konfigurationsdaten des PCI-Gerätes über das Register CONFIG_DATA (0xcfch) auslesen. Die PCI Implementierung in JX stellt hierzu bereits geeignete Methoden zur Verfügung, wodurch der Zugriff auf die Geräteinformationen für den Programmierer sehr abstrakt möglich ist.

### 2.2 PCMCIA

Bei PCMCIA handelt es sich eigentlich um eine Organisation, die Personal Computer Memory Card Inteface Association, welche im Jahre 1989 gegründet wurde und sich mit der Standardisierung vom Memory Cards beschäftigt - später wurden auch I/O-Karten mit hinzugenommen.

Es lassen sich, bedingt durch die Entwicklung im Laufe der Jahre, zwei Kartentypen unterscheiden, PC Cards und die weiterentwickelten CardBus-Karten. Beide Kartentypen sind mit heutigen PCMCIA-Sockets verwendbar, jedoch sind die verwendeten Zugriffsmechanismen sehr unterschiedlich.

Nachfolgend einige Kennzahlen zur Verdeutlichung der Unterschiede der beiden Kartentypen.

**Tabelle 1 Unterschiede zwischen PCMCIA und Card-Bus**

| Parameter | PCMCIA | Card-Bus |
|---|---|---|
| Schnittstelle | 68 Pins | 68 Pins |
| Takt (max.) | asynchron | 33 MHz |
| Bandbreite | 8-20 MByte/s | 132 MBytes/s |
| Datenbusbreite | 16 Bit | 32 Bit |
| Adressierung | 26 Bit | 32 Bit |
| Spannung | 5V, 3,3V | 3,3V |
| Interruptkanäle | 1 | 1 |
| Konfiguration | Attributspeicher | Configuration Space |
| Karteninformationen | CIS | Configuration Space |

Im Rahmen von AKBPII wurde im ersten Schritt die Treiberunterstützung zur Verwendung der PCMCIA-Sockets implementiert. Die Sockets werden vom PCI-Bus direkt als PCI-Geräte gefunden und können über den PCI-typischen “Configuration Space” (siehe Bild 3 auf Seite 3) eingestellt werden.

Laut PCMCIA-Standard [PCMCIA] bieten sich 2 Möglichkeiten zum Zugriff auf die Steckkarten an, Memory Mapping beziehungsweise I/O Ports und die Kartenregister.

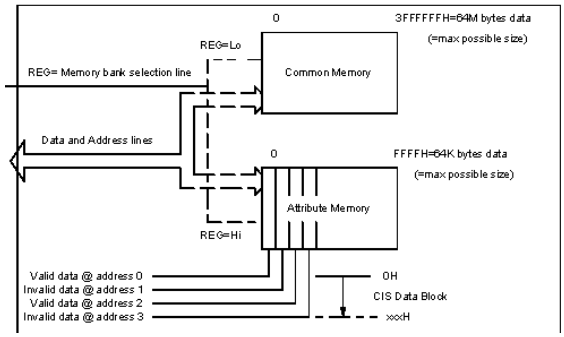
In der ursprünglichen Form des PCMCIA Standards wurde Memory Mapping angewandt, was jedoch die Verwendung spezieller Treiber nötig machte, da die normalen Gerätetreiber, zum Beispiel der Treiber der seriellen Schnittstelle, nicht über Memory Mapping ansprechbar war.

Aus diesem Grund wurde in der Version 2.0 des PCMCIA-Standards die Möglichkeit geschaffen, I/O-Adressbereiche der Steckkarten in den I/O Systemadressbereich einzublenden, wodurch die Wiederverwendbarkeit bereits vorhandener Treiber ermöglicht wurde.

Für den weitern Zugriff auf die Karteninformationen und -funktionen der in diesen Sockets eingelegten Steckkarten, welche zur Identifizierung der Karte dienen, bieten die beiden Kartentypen unterschiedliche Konzepte an (siehe Tabelle 1 auf Seite 2).

Bei PC Cards liegen diese Informationen in der CIS (Card Information Structure) im Attribute Memory (Attributspeicher) der Karte. PC-Cards verwenden laut Spezifikation zwei voneinander getrennte Speicher, den Attributspeicher, in welchem Konfigurationsinformationen hinterlegt sind und das Common Memory, in welchem die eigentlichen Nutzdaten der Steckkarten vorzufinden sind. Das nachfolgende Bild 2 soll hierzu einen kurzen Überblick geben.

**Bild 2 Aufbau einer PCCard**



Die im Attributspeicher verfügbare CIS ist eine verkettete Liste von Tupeln, wobei ein Tupel eine maximale Länge von 255 Bytes aufweisen kann und einen definierten Aufbau hat. Im ersten Byte steht der TPL-CODE, eine eindeutige ID zur Identifizierung der nachfolgenden Information. Das zweite Byte (TPL-LINK) legt die Länge der nachfolgenden Nutzinformationen fest, fungiert also als eine Art Zeiger auf das nachfolgende Tupel. Im Rahmen des Auslesens der CIS werden Informationen über den E/A- bzw. Speicherbedarf der Karte, die benötigte Spannung und die Interruptanforderungen ausgelsen.

Card-Bus-Karten, welche zeitlich gesehen nach dem PCI-Bus entwickelt wurden, sind PCI-Geräten sehr ähnlich und werden daher vom PCI-Bus nach erfolgter Initialisierung auch als solche erkannt. Die Konfiguration der Card-Bus-Karten erfolgt PCI-typisch über den “Configuration Space”, wodurch die weitere Handhabung der Karte mit bereits vorhandenen Mitteln für PCI weitestgehend möglich wird.

**Bild 3 Configuration Space des Texas Instruments 1520 Sockets**

**PCI Configuration Registers (Functions 0 and 1)**

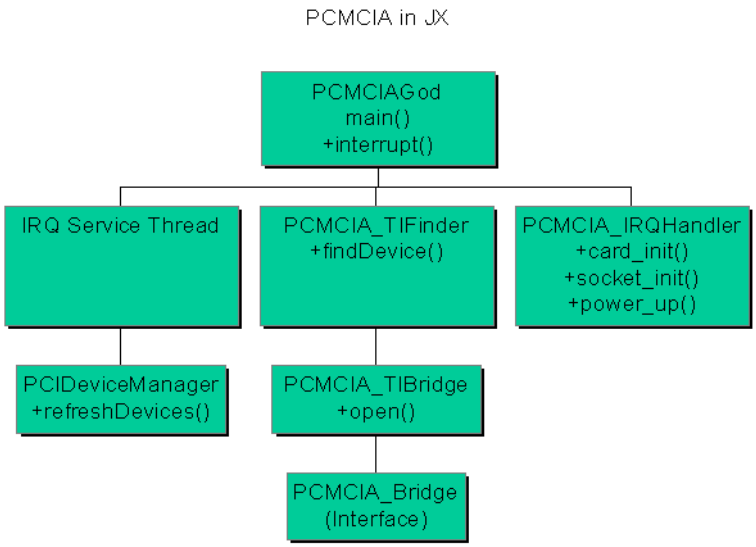
| REGISTER NAME | | | | OFFSET |
|---|---|---|---|---|
| Device ID | | Vendor ID | | 00h |
| Status | | Command | | 04h |
| Class code | | | Revision ID | 08h |
| BIST | Header type | Latency timer | Cache line size | 0Ch |
| CardBus socket/ExCA base address | | | | 10h |
| Secondary status | | Reserved | Capability pointer | 14h |
| CardBus latency timer | Subordinate bus number | CardBus bus number | PCI bus number | 18h |
| CardBus Memory base register 0 | | | | 1Ch |
| CardBus Memory limit register 0 | | | | 20h |
| CardBus Memory base register 1 | | | | 24h |
| CardBus Memory limit register 1 | | | | 28h |
| CardBus I/O base register 0 | | | | 2Ch |
| CardBus I/O limit register 0 | | | | 30h |
| CardBus I/O base register 1 | | | | 34h |
| CardBus I/O limit register 1 | | | | 38h |
| Bridge control | | Interrupt pin | Interrupt line | 3Ch |
| Subsystem ID | | Subsystem vendor ID | | 40h |
| PC Card 16-bit I/F legacy-mode base address | | | | 44h |
| Reserved | | | | 48h–7Ch |
| System control | | | | 80h |
| Reserved | | | | 84h–88h |
| Multifunction routing | | | | 8Ch |
| Diagnostic | Device control | Card control | Retry status | 90h |
| Reserved | | | | 94h–9Ch |
| Power-management capabilities | | Next-item pointer | Capability ID | A0h |
| Power-management data | Power-management control/status bridge support extensions | Power-management control/status | | A4h |
| General-purpose event enable | | General-purpose event status | | A8h |
| General-purpose output | | General-purpose input | | ACh |
| Serial bus control/status | Serial bus slave address | Serial bus index | Serial bus data | B0h |
| Reserved | | | | B4h–FCh |

# 4 Implementierung des PCMCIA-Socket-Treibers

Im Verlaufe der Implementierung der PCMCIA-Unterstützung für JX wurde, wie bereits geschreiben, zuerst ein Treiber für die PCMCIA-Sockets selbst implementiert. Nach Abschluss dieser Arbeiten wurde das Hauptaugenmerk auf die Implementierung der CardBus-Steckarten Unterstüztung gelegt. In JX ist bereits eine Vielzahl an PCI-Gerätetreibern vorhanden, so zum Beispiel für die 3COM905 Netzwerkkarte, welche für die Verwendung mit PCMCIA-CardBus Geräten geeignet sind.

Nachfolgend soll die Implentierung der beiden Treiberschichten (Socket und CardBus) genauer besprochen werden. Aus diesem Grunde folgt eine kurze graphische Zusammenstellung der verwendetne Klassen mit den wichtigen Methoden dieser Klassen zur besseren Veranschaulichung (siehe Bild 5).

**Bild 5 Aufbau der Treiberimplementierung in JX**




## 4.1 PCMCIAGod

Der PCMCIAGod repräsentiert den PCMCIA-Bus im Allgemeinen. Beim Starten werden die Geräte-Finder für die unterstützten PCMCIA-Sockets gestartet. Weiterhin führt der PCMCIAGod die grundlegende Initialisierung der Sockets durch und ist für die Interruptbehandlung verantwortlich. Durch Auslesen der Kartenregister kann die Quelle und die Art des Interrupts ermittelt werden, wobei die Behandlung von CardChange-Interrupts in einen externen Dienst-Thread ausgelagert ist, welcher bei Bedarf deblockiert wird.

## 4.2 PCMCIA_TIFinder

Der PCI-Bus sucht beim Start nach verfügbaren PCI-Geräten und speichert die gefunden Geräte intern ab. Gerätetreiber können diese Informationen über definierte Schnittstellen abfragen und so die Existenz einer unterstützten Gerätes im System überprüfen. Dieses Vorgehen wurde mit Hilfe eines speziellen Finders für Texas Instruments PCMCIA-Brücken ebenfalls gewählt, um die im Computer verfügbaren und unterstützten Geräte zu finden.

## 4.3 PCMCIA_Bridge

Die Klasse PCMCIA_Bridge stellt eine Schnittstellendeklaration für die Anbindung von PCMCIA-Sockets in JX zur Verwendung, welche die Einbindung von weiteren Chips verschiedener Hersteller ermöglicht.

### 4.3.1 PCMCIA_TIBridge

Bei der PCMCIA_TIBridge handelt es sich um die Implementierung der Texas Instruments PCMCIA-Bridges. Die Klasse stellt verschiedene Methoden zur Initialisierung und Konfiguration der Sockets zur Verfügung und dient vor allem zur Kapselung des den PCMCIA-Socket respräsentierenden PCIDevices.

### 4.3.2 Interrupt Handling als Service Thread

Der PCMCIAGod erzeugt beim Starten einen Thread, welcher für die asynchrone Bearbeitung von Interrupts und das dynamische Nachladen der Gerätetreiber für neu gefundene Steckkarten verantwortlich ist. Der Service Thread blockiert sich an einer AtomicVariable, welche bei auftreten eines Interrupts entsprechend dekrementiert wird. Anschließend ruft der Thread über ein Portal die refreshDevices() Methode des PCIDeviceManager auf, wodurch der PCI-Bus auf neue Geräte hin durchsucht wird und die entsprechenden Treiber für neue Geräte nachgeladen werden.

# 5 Initialisierung von Cardbus-Karten

Der zentrale Punkt der Treiberprogrammierung für den PCMCIA-Socket anhand des Texas Instruments Chipsatzes ist die richtige Einstellung der Steuerregister. Hierbei ist die Reihenfolge der Befehle von großer Bedeutung. Der prinzipielle Vorgang ist wie folgt: Command und Bridge Register einstellen, anschließend muss die benötigte Versorgungsspannung der eingesetzten Steckkarte anhand der Statusregister ermittelt werden. Im weitern Verlauf kann dann der Socket mit Strom versorgt werden. Nach Abschluss dieses Vorganges gibt die Karte in READY-Signal aus, welches wiederum über die Statusregister des Sockets ausgelesen werden kann. Nun erfolgt ein Reset der Karte. Anschließen kann die Karte als PCI-Geräte erkannt und angesprochen werden. Im weiteren Verlauf kann der Gerätetreiber basierend auf der VendorID und DeviceID ermittelt und geladen werden.

# 6 Zusammenfassung

Im Rahmen der gestellten Aufgabe wurde ein Konzept zur Implementierung der PCMCIA-Unterstützung in JX erstellt und in Form einer entsprechenden Klassenhierarchie in Java umgesetzt.

Desweiteren wurde ein Treiber für den Texas Instruments 1250/1520 Chipsatz programmiert.

Durch die Erweiterung des Treibers für den PCI-Bus und das Treibermanagement durch Martin Mitzlaff werden nun für die PCMCIA Implementierung grundlegende Fähigkeiten des PCI-Buddes implementiert, wodurch eine hotplugfähige PCMCIA-Unterstützung in JX umgesetzt werden kann.

# 7 Literaturverzeichnis


TI1520. Texas Instruments, PCI1520/PCI1520I GHK/PDV PC Card Controllers - Data Manual, 2003, http://www-s.ti.com/sc/ds/pci1520.pdf

PCHWB03. Messmer, Dembowski: PC Hardwarebuch (7. Auflage), Addison-Wesley, Bosten, Juni 2003, ISBN: 3-8273-2014-3, Seiten 999-1014

PCMCIA. Webseite der PCMCIA: http://www.pcmcia.org