Bootvorgang bei Symbian OS

Martin Steininger

TH Deggendorf

martin.steininger@stud.th-deg.de

13. November 2015

Inhalt

Geschichte (1)

- Ursprünge in 32-BIT-EPOC-Plattform von Psion
- 1998: neu gegründetes Konsortium firmiert unter dem Namen *Symbian* durch Mobilfunkunternehmen Ericsson, Motorola, Nokia und Psion übernimmt Weiterentwicklung
- Später: weitere Unternehmen setzen Symbian auf ihren Mobiltelefonen ein (unter anderem Samsung)

Geschichte (2)

- 2008: Nokia übernimmt die Symbian Ltd. vollständig und überführt sie sukzessive in eine gemeinnützige Organsiation
- 2008: Nokia erwirbt alle Rechte an Symbian und überträgt diese an die Symbian Foundation
- 2010: Symbian wird Open Source
- 2012: Einstellung der Weiterentwicklung

Oberfläche

Abbildung: Symbian OS mit Touchoberfläche

Abbildung: Symbian OS ohne Touch

Übersicht (1)

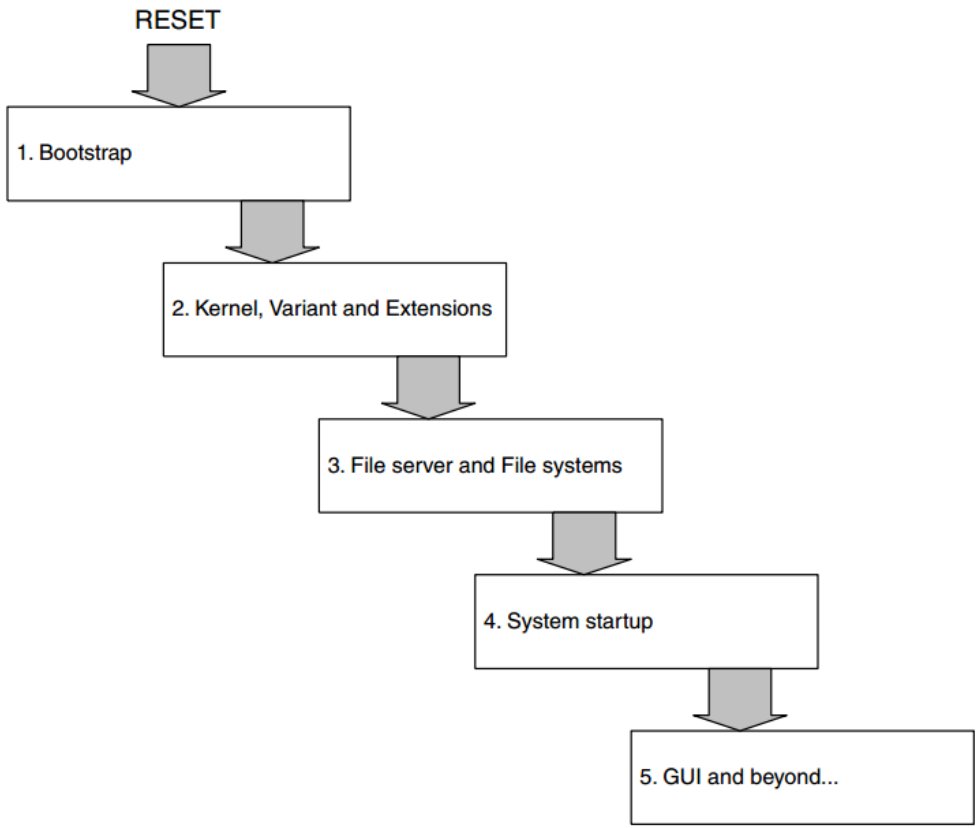
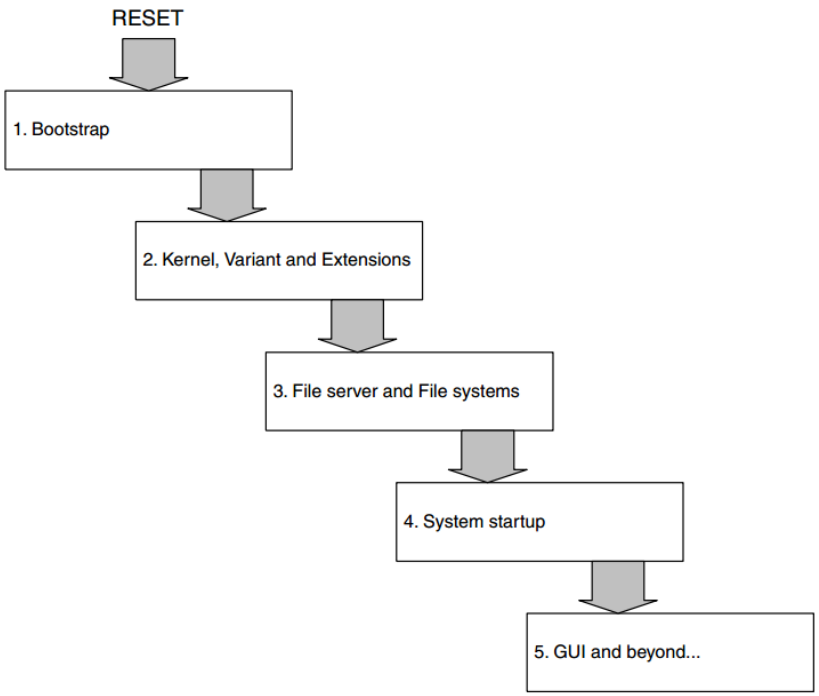


Abbildung: Stufen beim Symbian OS Bootvorgang

Übersicht (2)



1. Bootstrap
2. Kernel
3. Filesystem
4. Systemstart
5. GUI

Bootstrap (1)

- das Einschalten des Telefons löst einen kompletten Hardwarereset aus
- Start von *Bootstrap* (deutsch etwa: Startprogramm) als erste Software
- Ausführungsumgebung ist sehr rudimentär und primitiv
  - Memory Management Unit (MMU) ist deaktiviert
  - CPU und Speicher laufen mit sicheren und niedrigen Geschwindigkeiten
  - CPU verwendet nur ein Register: Programmzähler (Start bei Adresse 0)

Bootstrap (2)

- Bootstrap verwendet eine geordnete Struktur (*superpage*), die Informationen zwischen bootstrap und Kernel transportiert
  - Adressen und Größe der Speicherblöcke und -cluster (z.B. Rootverzeichnis)
  - verschiedene andere Werte, die vom Startprogramm errechnet und bereitgestellt werden
  - die Superpage liegt immer im gleichen Speicherbereich, sodass Kernel und Bootstrap immer zugreifen können
- Bootstrap generiert und initiiert den RAM page allocator (RAM-Speicherzuordnungstabelle)

Bootstrap (3)

- Allozieren, säubern und Mapping des primary page directory
- Mapping des ROM und anderer Datenstrukturen
- Mapping der Hardware `I/O`
- MMU wird auf aktiv gesetzt und ermöglicht virtual addressing
- Kernel Ausführung wird vorbereitet:
  - Initial Thread Stack wird alloziert und anhand der Speichergröße im ROM erstellt
  - Ähnlich werden statische Daten des Kernel initalisiert
- Ausführung des Kernels

Kernel (1)

- Ab diesem Punkt läuft die CPU (zumindest kurzzeitig) auf maximaler Geschwindigkeit
- Primitive und rudimentäre Umgebung kann nun C++-Code ausführen
- Speicherumgebung ist jedoch immer noch sehr rudimentär und bietet lediglich einen Ausführungspfad
- CPU initialisiert alle Ausführungsmodi, jedoch nur ein gemeinsamer Arbeitsstack
  - Neu: Exceptionhandling und Diagnoseerstellung
  - dadurch treten keine schwer zu debuggenden Fehler mehr auf

Kernel (2) und init-0

- Kernel lädt C++ Konstruktoren für statische Kernelobjekte
- Initialisierung durch Initialisierungsroutinen *init-0* bis *init-3*

init-1

- Initiiert statische Datenobjekte für alle *Board support packages*
  - Standard Befehlssätze der Hard- und Software
- außerdem: Initialisierung der Variante (abhängig von den geladenen Kernelerweiterungen)

init-1

- Board support packages (BSP) sind initialisiert und verwendbar
- Kernel und BSP Objekte werden der Reihe nach vorbereitet
  - MMU und Cache Management Objekte
  - Coprocessor Management
  - Interrupt Dispatcher (Interrupt Handling)
  - Exception Mode Stacks für IRQ, FIQ (Physikalische Hardwareadressen einzelner Komponenten)
- Freier Speicher wird alloziert, dynamisch jedoch noch nicht erweiterbar (Lokale Dateien werden auf Default-Werte gesetzt, Threading wird eingerichtet)
- Scheduler wird aktiviert

init-2

- Finale Initialisierung des Speichermanagers → gesamter Adressraum steht zur Verfügung
- Falls „warm reset“: Wiederherstellung aller im RAM gespeicherten Inhalte, die „überlebt“ haben
- Finale Initialisierung des Coprocessors erzeugt eine Threadumgebung mit gespeicherten Coprocessorzuständen
- fehlende Kernelressourcenmanagement Maschinerie wird gestartet
  - Dynamische Speicherallokation wird ermöglicht
  - Debuggerinterface, publish and subscribe, Energiemodell und -steuerung, Code Management wird endgültig initialisiert und hat volle Funktionalität
- Supervisor Thread wird gestartet

init-3

- Supervisor Thread startet weitere Kernelservices
  - Hardware Abstraction Layer → Abstraktion des Kernels bzw. gesamten Betriebssystems von der Hardware
  - event queue → sukzessives Abarbeiten gestarteter Tasks
  - RAM drive chunk → Arbeitsspeicher wird vollständig adressierbar
  - tick and second-timer System wird gestartet → zeitabhängiges Abarbeiten von Threads, Tasks etc.
- Am Ende von init-3 wird der Start des Fileservers bzw. -systems angestoßen

Filesystem (1)

- Mikrokernel läuft, jedoch keine Möglichkeit persistente Daten im read/write Flash zu schreiben oder Daten zu lesen
- Sekundärthread wird gestartet, um read / write Locks und Deadlocks zu vermeiden bzw. verhindern
- ein erstes Dateisystem (XIP ROM) wird gemountet, was dann den Fileserverservice ermöglicht
- Fileserver erzeugt Dateinamencache und schließt Initialisierung der lokalen Laufwerkskollektion ab

Filesystem (2)

- Ausführbare Dateien im XIP ROM installieren lokale und physikalische Medientreiber
- read / write Filesystem gänzlich verfügbar, HAL Einstellungen werden auf dem internen Laufwerk gefunden und wiederhergestellt (Spracheinstellungen!)
- Kompletter Kernel, Benutzerbibliothek, Dateisystemservices sind vollständig initialisiert $\rightarrow$ Systemstart wird ausgeführt

Systemstart & Gui

- Framework für den normalen Betrieb → Starten und verwalten aller Services
- Telefonfunktion wird als einer der ersten Services gestartet
- Window Server wird gestartet → Zugriff auf Display, Tastatur, Touchscreen

- Start der GUI, die je nach Hersteller eine andere Optik aufweisen kann

verschiedene GUIs

Abbildung: Auswahl einiger GUIs, durch Mobilfunkanbieter, Hersteller und Benutzer angepasst

Quellen


Jane Sales, Andrew Thoelke and Carlos Freitas with Jon Coppeard (2005)
Symbian OS Internals - Real-time Kernel Programming


Andrew S. Tannenbaum (2003)
Moderne Betriebssysteme (Pearson Studium-IT)


P. Weisberg, Y. Wiseman
Using 4KB Page Size for Virtual Memory is Obsolete


Hanna Tam (2010)
Symbian. Anwendungs- und Spieleentwicklung für S60v3, S60v5 und Symbian^3


Christoph Tenbergen, Holger Heller
Embedded-Linux-Systeme schnell und einfach konfigurieren
http://www.elektronikpraxis.vogel.de/softwareengineering/betriebsysteme/articles/360445/index2.html, 14.2.2012

Vielen Dank