

Die Rettung der "Alten Welt"

Anwendungs-Migrationen und -Portierungen von Legacy-Systemen in die „offene Welt“

Teil III - Anwendungs-Portierung und -Modernisierung Systeme mit Unix-Derivaten

Verfasser: Andreas Maier
OrgaBIT
Organisation • Beratung • IT

Datum: 10.09.2008

Nachdem wir im Teil I die Begriffe Migration und Portierung erläuterten und die möglichen Migrationsstrategien vorstellten, soll hier nun näher auf den von uns präferierten Migrationsweg einer Software-Portierung mit einhergehender Modernisierung bzw. teilweisem Reengineering der Anwendung eingegangen werden.

Nochmals zur Erinnerung: eine Portierung ist die physische Übernahme von Anwendungsprogrammen und -daten auf eine neue Betriebssystem-Umgebung und die Konvertierung des existierenden Sourcecodes und der Nutzdaten damit diese in der neuen Umgebung fehlerfrei genutzt werden können.

Eine Portierung bietet sich grundsätzlich dann an, wenn die betreffende Anwendung auf die Prozesse des Unternehmens optimal zugeschnitten ist, die Anwendung als geschäftskritisch eingestuft werden kann und natürlich auch durch Individual-Programmierung entstanden ist.

Natürlich stellt sich immer die Frage, auf welche Hardware-/Betriebssystem-Kombination heute normalerweise eine Anwendungs-Migration geplant und durchgeführt wird: aufgrund der Offenheit, der freien Verfügbarkeit, der Unterstützung durch alle großen IT-Hersteller und damit des weltweiten Supports und nicht zuletzt aufgrund der Verfügbarkeit einer immens großen Anzahl jeglicher Art von Middleware, Werkzeugen und Bibliotheken wird als Zielplattform für eine Anwendungsportierung heute typischerweise immer ein Linux-basierendes System gewählt. Bei einigen Portierungen kommen auch die vom jeweiligen Hersteller angebotenen Systeme mit einem der Unix-Derivate zum Einsatz, weil diese Systeme teilweise schon Erweiterungen beinhalten, welche den Portierungsvorgang einfacher machen können.

Da für die in Teil I erwähnten betroffenen Typen von Systemarchitekturen sehr unterschiedliche Portierungsansätze gelten und die Fragestellungen und Vorgehensweisen während einer Portierung sehr unterschiedlich ausfallen, werden in diesem Teil nur die Klasse der „RISC-Systeme mit Unix-Derivat“ abgehandelt. In Teil II gehen wir detailliert auf die Vorgehensweisen bei Anwendungsportierungen von „proprietären Systemen mit herstellereigenem Betriebssystem“ ein.

RISC-Systeme mit Unix-Derivaten

Im Gegensatz zu den unter I. diskutierten proprietären Rechnersystemen mit herstellereigenen Betriebssystemen soll hier näher auf Unix-basierende Systeme eingegangen werden.

Man spricht zwar hier von den „offenen Systemen“ (open systems), da sie im Allgemeinen den wesentlichen Standards angepasst sind (POSIX, ISO, OSF, Open Group, etc.), jedoch sind auch diese Systeme mit weiteren herstellereigenen Funktionalitäten ausgestattet, sofern für spezielle Anforderungen keine entsprechenden Standards zur Verfügung standen – auch hier ist ein „Vendor Lock In“ gegeben.

Ein paar Beispiele für diese Klasse von Systeme sind

Hersteller	Plattform	Architektur	Unix-Derivat / Basis
HP	HP9000	HP-PA (Precision Architecture)	HP-UX AT&T System V
IBM	System p (pSeries)	Power Architecture	AIX Open Group Single UNIX Specification, Version 3
Sun	Sun SPARC Enterprise Sun Fire	SPARC / UltraSPARC	SunOS / Solaris AT&T System V
HP (ehem. DEC)	Alpha Systems	Alpha	OSF/1 / True64 Unix AT&T System V

Diese Systeme haben alle gemeinsam, dass sie zum aktuellen Produktangebot des jeweiligen Herstellers gehören und von diesen auch weiterentwickelt, verkauft und unterstützt werden. Die zugrunde liegenden Architekturen wurden aber auch seit Einführung teilweise mehrfach geändert und weiterentwickelt (32 Bit → 64 Bit, Prozessor-Befehlssatz, etc.) und diese Änderungen hatten zur Folge, dass die darauf laufenden Anwendungen nur durch Neu-Übersetzung diese neuen Entwicklungen auch nutzen konnten.

Unter einem Unix-Derivat implementierte Individual-Anwendungen werden typischerweise immer in Richtung Linux portiert, da dies als ein natürlicher Schritt betrachtet werden kann und da auf Grund der Ähnlichkeiten und der gemeinsamen Standards dieser Betriebssysteme weniger Probleme bei der Portierung zu erwarten sind. Darüber hinaus sind Server-Systeme mit Linux Betriebssystem inzwischen so ausgereift, dass damit ein reibungsloser IT-Betrieb gewährleistet werden kann und auch geschäftskritische Anwendungen darauf eingesetzt werden können - Linux hat Einzug in die Business-IT genommen.

In vielen Fällen erweist sich eine Unix zu Linux Portierung als sehr unkompliziert - die zu portierende Anwendung wird auf der neuen Plattform ohne große Änderungen neu kompiliert und anschließend neu validiert. In anderen Fällen gestaltet sich der Prozess jedoch wesentlich schwieriger, speziell, wenn sehr große Anwendungen auf älteren, proprietären Unix-Derivaten unter Nutzung herstellerspezifischer Middleware (Compiler, Datenbanken, Forms Management, Command Language, etc.) entwickelt wurden und die genutzte Middleware nicht auf der neuen Plattform verfügbar ist oder nicht mehr den gängigen Standards entspricht.

Linux-Distributionen

Trotz der Vielzahl von heute verfügbaren Linux-Distributionen, gestalten sich grundsätzlich Unix zu Linux Portierungen immer ähnlich, da all diese Distributionen letztendlich auf der GNU/Linux Definition aufbauen und die für eine Portierung notwendigen Entwicklungswerkzeuge und Tools (Compiler, Debugger, System-Bibliotheken, etc.) unabhängig von

der jeweiligen Distribution sind. Im Wesentlichen unterscheiden sich die jeweiligen Distributionen hauptsächlich in den mitgelieferten Software-Paketen, der System-Administration und -Wartung und dem durch den Hersteller angebotenen Support. Für kommerzielle Zwecke sind vor allem die Linux-Distributionen von Novell (SLES/SuSE Linux Enterprise Server), Red Hat (Enterprise Linux) und Debian (GNU/Linux) geeignet:

Distribution	unterstützte Architekturen	Kernel (08.2008)
Novell SuSE Linux Enterprise Server	Intel + AMD x86/x86-64 Intel IA-64 / Itanium 2 IBM PowerPC IBM S/390 und System z	2.6.16
Red Hat Enterprise Linux	Intel + AMD x86/x86-64 Intel IA-64 / Itanium2 IBM PowerPC IBM S/390 und System z	2.6.18
Debian GNU/Linux	Alpha ARM HP PA-RISC Intel + AMD x86/x86-64 Intel IA-64 / Itanium 2 MIPS (big endian) MIPS (little endian) IBM PowerPC IBM S/390 und System z SPARC	2.6.18

Sofern verfügbar, möglich und sinnvoll werden bei einer Unix zu Linux Portierung immer die mit der jeweiligen Linux-Distribution mitgelieferten Compiler und Tools verwendet. Neben den üblichen Unix- und Linux-Entwicklungswerkzeugen wie *make*, *awk*, *sed*, etc. sind dies vor allem *gcc* (GNU Compiler Collection - C/C++, Objective C/C++, Java, Ada und Fortran) und der *gdb* Debugger. Erfahrungsgemäß erfüllt der gcc C/C++ Compiler alle Anforderungen bei Softwareportierungen, sofern aber Compiler für andere Programmiersprachen benötigt werden (z.B. Cobol, Fortran, etc.) oder auch die Funktionalität der in gcc beinhalteten Compiler-Komponente bei komplexen Anwendungen nicht ausreicht, werden auch kommerziell verfügbare Compiler eingesetzt.

Portierungswerkzeuge und Portierungsprojekte

Erfahrungsgemäß lassen sich Anwendungen, welche in C und C++ entwickelt wurden, mit eventuell notwendigen geringen Anpassungen am Source-Code mit dem gcc Compiler auf Linux relativ schnell fehlerfrei übersetzen. Allerdings treten danach während der ersten Tests häufig Laufzeitfehler oder Programmabbrüche auf. Dies liegt vor allem daran, dass die genutzten Systemroutinen normalerweise zwar die gleichen Aufrufparameter nutzen (da auf Standards basierend), die entsprechenden Routinen aber dann unterschiedlich auf den jeweiligen Plattformen implementiert wurden. Als Beispiel seien hier die *mem*()* Routinen (*memcpy()*, *memcmp()*, etc.) genannt, welche auf allen Systemen mit den gleichen Parametern definiert sind (POSIX.1-2001), aber in der Linux-Variante das Über-

geben eines NULL-Pointers mit einem Programmabbruch quittiert wird, während z.B. unter HP-UX die Übergabe von NULL-Pointern fehlertolerant abgehandelt wird.

Unix zu Linux Portierungen anderer Programmiersprachen (Fortran, Cobol, etc.) sind typischerweise immer mit mehr Aufwand verbunden, da es für diese Sprachen zwar Standards gibt, diese Standards aber über die Jahre immer weiterentwickelt und teilweise auch geändert wurden und die diversen Hersteller auch Zusatzfunktionalitäten, welche keinem Standard entsprechen (z.B. Compiler-Optionen, Behandlung von Shared Memory), in die Compiler implementierten.

Desweiteren sind bei einer Portierung immer dann Eingriffe und zusätzlicher Aufwand notwendig, wenn in der zu portierenden Anwendung herstellerspezifische Systemroutinen genutzt werden. So implementierte zum Beispiel Hewlett-Packard im Kernel des HP-UX Betriebssystems auch Echtzeit-Funktionalitäten und das entsprechenden API dazu, für welche es ursprünglich keine Standards-Definition gab. Werden diese Art von Funktionalitäten in einer Anwendung genutzt, sind Anpassungen oder Zusatzentwicklungen notwendig.

Ein Unix zu Linux Portierungsprojekt beinhaltet typischerweise folgenden Tätigkeiten und Dienstleistungen:

- » Analyse der zu portierenden Anwendung und Aufwandsabschätzung
- » Installation und Konfiguration der Linux-Distribution, gegebenenfalls Zurverfügungstellung von Hardware während der Portierung
- » Installation und Konfiguration der notwendigen und sinnvollen Entwicklungswerkzeuge (Compiler, Debugger, etc.)
- » Aufbau einer geeigneten Entwicklungsumgebung
- » Generierung / Anpassung von Makefiles
- » Source Code Konverter
- » Automatische Generierung von Prototype-Definitionen für Anwendungsroutinen
- » Entwicklung von Emulationsroutinen für herstellerspezifische Systemroutinen
- » Entwicklung von fehlertoleranten Zwischenschichten für Systemroutinen
- » Durchführung von Modernisierungs-Maßnahmen während oder nach der Portierung, Entwicklung und Anpassung entsprechender Schnittstellen
- » Schulungen bzgl. Linux Entwicklungswerkzeugen und Linux generell

Fallbeispiel für eine Unix zu Linux Portierung

Ein großes, namhaftes Maschinenbauunternehmen beauftragte Anfang der 90er Jahre die Firma Siemens, für ein neu errichtetes Logistik- und Servicezentrum eine geeignete Logistik-Anwendung zu entwickeln. Als Basis für die Entwicklung wurde seinerzeit die HP9000-Plattform von Hewlett-Packard mit dem Unix-Derivat HP-UX ausgewählt, da Hewlett-Packard zu dieser Zeit Marktführer in diesem Segment war, die HP9000-Serverfamilie als sehr leistungsfähig und skalierbar galt und sowohl Hardware als auch Betriebssystem sehr robust und zuverlässig waren. Die Anwendungen wurden in der Programmiersprache C entwickelt und als Datenbanksystem wurde Oracle verwendet.

Im Rahmen einer strategische Entscheidung des Unternehmens sollen alle Unix-basierenden Systeme im Konzern nach und nach durch Linux-Systeme ersetzt werden.

Die Gründe für diese Entscheidung sind naheliegend: die bisher im Einsatz befindlichen HP9000-Server sind am Ende ihres Lebenszyklus und Hardware auf Intel-Basis mit Linux-Betriebssystem ist wesentlich kostengünstiger (TCO: Anschaffung, Leistung/\$, Wartung, Betrieb). So wurde vom Unternehmen für das Jahr 2008 die Portierung der Logistik-Anwendungen auf Linux geplant und und auch schon ein Termin für den Produktiveinsatz der portierten Anwendungen definiert.

Als Basis wurde nun die Linux-Distribution von RedHat gewählt, als C-Compiler sollte der standarmäßig mit Linux ausgelieferte gcc (C-Compiler der GNU Compiler Collection) verwendet werden. Oracle Datenbank- und Maskensystem kamen in der Version 10.1 zum Einsatz.

Bei den ersten vom Unternehmen selbst durchgeführten Tests wurde festgestellt, dass der eigentliche Portierungsvorgang zwar relativ einfach durchzuführen ist, allerdings die Lauffähigkeit der portierten Programme auf dem Linux-System aufgrund vieler Programmabbrüche nicht gegeben war. Speziell bei der Übergabe von falschen bzw. nicht gesetzten Parametern in Funktionen terminierten die Programme mit den "segmentation violation" (SIGSEGV) bzw. "bus error" (SIGBUS) Signals. Das Unternehmen beauftragte daraufhin nun die Firma OrgaBIT, als Spezialist für Anwendungs-Migrationen bekannt, zu untersuchen, warum genau die Programme terminierten und eine einfache Lösung zu implementieren, sodass die Gründe für die Programmabbrüche beseitigt bzw. umgangen werden konnten.

Die ersten Untersuchungen ergaben folgende hauptsächlichen Gründe für die Terminierung der portierten Programme - die von OrgaBIT implementierte Lösung ist aus der rechten Spalte ersichtlich:

	Problem	Lösung
1.	Übergabe von falschen (falscher Typ), unvollständigen (zu wenig) oder nicht gesetzten (NULL-Pointer) Parametern in die häufig verwendete Trace-Funktion. Diese Funktion schreibt Programm-Trace-Meldungen in entsprechende Trace-Files. Der Inhalt der Meldungen wird über einen Format-String und eine variable Anzahl von Parametern an die betreffende Funktion übergeben.	Nutzung einer entsprechenden Compiler-Option des gcc C-Compilers, um die fehlerhaften Stellen durch eine Funktionsprototypen-Deklaration während der Übersetzung herauszufiltern: <pre>int trace (const arg1, char *arg2, ...) __attribute__ ((format (printf, 2,3)));</pre>
2.	Übergabe von Strukturen in Funktionen „per Value“ anstatt „per Reference“, Übergabe von nicht Typ-konformen Parametern in Funktionen.	Automatische Generierung von Funktionsprototypen-Deklarationen (Mitteilung an den Compiler, welche Funktion mit welchen Parametern und Datentypen aufgerufen werden muss) für alle Anwendungs-Funktionen und Zusteuerung dieser als Header-Files für den Übersetzungsvorgang. So gibt der Compiler für nicht Typ-konforme Übergabeparameter entsprechende Fehlermeldungen aus.
3.	Übergabe von NULL-Pointern in die mem*() und str*() libc-Funktionen.	Implementierung einer fehlertoleranten Zwischenschicht für die betroffenen Funktionen

		und automatische Ersetzung der Funktionsaufrufe im Anwendungs-Sourcecode während des Übersetzungsvorganges.
4.	Wert-Zuweisungen von Strukturen mit nicht korrekten Struktur-Pointern (NULL bzw. geschützte Adressen).	Hier war es nicht möglich eine automatische Lösung zu implementieren - die entsprechenden Stellen mussten händisch abgeändert werden.

Aus der Tatsache, dass diese Fehler unter HP-UX nie aufgefallen waren und auch nicht zu Programmabbrüchen führten ist ersichtlich, dass der HP-UX C-Compiler und die HP-UX libc-Bibliothek mit dieser Art von Fehlern wesentlich kulanter umgehen und diese auch teilweise automatisch korrigieren.

Insgesamt waren die erwähnten notwendigen Maßnahmen, die Erstellung von entsprechenden Analyse-, Konvertierungs- und Generierungsscripts und die anschließend notwendigen Änderungen am Anwendungs-Sourcecode mit einem Aufwand von ca. 6 Wochen verbunden, nach ca. 4 Wochen intensiven Tests konnten die auf Linux portierten Anwendungen erfolgreich in den Produktivbetrieb übernommen werden.

© ® Alle Markennamen, Warenzeichen und eingetragenen Warenzeichen der in diesem Dokument aufgeführten und erwähnten Produkte und Dienstleistungen sind Eigentum Ihrer rechtmäßigen Eigentümer und dienen hier nur der Beschreibung. Das Kopieren von Daten (jeglicher Art) aus diesem Dokument ist ohne schriftliche Zustimmung von OrgaBIT grundsätzlich untersagt.
