FreeBSD im Überblick

Christoph Herrmann

Berlin, den 13. Februar 2002

Einleitung

In diesem Vortrag soll FreeBSD als ganzes kurz vorgestellt werden. Es soll nach Möglichkeit eine Antwort auf die Frage: "Was ist FreeBSD?" gegeben werden. Dafür werde ich näher auf die Wurzeln von (Free)BSD, die Geschichte des Projektes, seine Organisation, sein Entwicklungsmodell und seine Ziele eingehen. Weiterhin werden einige FreeBSD "Specials" Erwähnung finden.

1 Die Geschichte von (Free)BSD

Die Geschichte von FreeBSD und aller anderen BSDs ist zu großen Teilen die Geschichte von Unix selbst. Sie beginnt als Ken Thompson und Dennis Ritchie 1969 auf einer ungenutzten PDP-7 anfangen ein neues Betriebsystem zu entwickeln, (das *Ur*-Unix *First Edition*) wobei viele der Ideen aus dem *Multics* Projekt kommen. Dieses System wird erst innerhalb der Bell Labs weiterentwickelt und kommt als *Sixth Edition* 1976 u.a. an die University of California at Berkely. Zur damaligen Zeit besaß dieses System mehrere herausragende Eigenschaften:

- UNIX war in einer high level Programmiersprache geschrieben (C)
- UNIX wurde mit dem Sourcecode verteilt
- UNIX lagen Konzepte zugrunde, die sonst nur auf sehr viel teurerer und größerer Hardware anzutreffen waren.

In Berkely wurde dieses UNIX System in den nächsten fast 20 Jahren entscheidend weiterentwickelt. Viele der Konzepte, die heute sowohl in allen Unixen, als auch in anderen Betriebssystemen anzutreffen sind, wurden in Berkely entwickelt.

Dies geschah auch damals schon nach einem Opensource typischen Modell. Es gab die CSRG (Computer Systems Research Group) an der Universität in Berkely, welche die Entwicklung steuerte, aber es waren Entwickler aus einem großen Teil der Welt (vornehmlich aus Universitäten und von der US Army), die das System weiterentwickelt haben.

Da die Grundlagen aber der Code von den Bell Labs war, brauchten Interessenten die eine Version von *BSD aus Berkely erwerben wollten immer auch eine Lizenz von AT&T. Um den in Berkely entwickelnte Code einer größeren Basis an Benutzern und Entwicklern zukommen zu lassen, faßte die Universität unabhängig von den *BSD Releases, ihre

Eigenentwicklungen, die frei von lizensiertem AT&T Code waren, als NET/1 (1989) und NET/2 (1991) zusammen und verbreitete sie kostenlos.

NET/2 war ein fast vollständiges Betriebsystem (4.3BSD) und die Grundlage für viele neue Projekte, z.Bsp. BSDIs BSD/OS und 386BSD. Aus dem 386BSD Projekt spalteten sich sowohl NetBSD als auch FreeBSD mit getrennten Zielsetzungen ab. Die Weiterentwicklung all dieser Projekte kam aber 1992 und 1993 fast vollständig zum erliegen, weil AT&T seine Urheberrechte an Teilen des Codes verletzt sah und gegen die University of California at Berkely prozessierte. Letztendlich einigten sich die Parteien mit einem Vergleich. Der Code des inzwischen releasten 4.4BSD wurde nochmal überarbeitet, einige Teile wurden neu geschrieben und das ganze als 4.4BSD Lite-1 releast. Daran hatte AT&T keine Ansprüche mehr.

4.4BSD Lite-1 ist u.a. die Grundlage des heutigen FreeBSD. Ein Ergebnis dieses Streites ist, das der heutige FreeBSD CVS Tree mit FreeBSD 2.0 (entspricht 4.4BSD Lite-1) beginnt und FreeBSD 1.* nicht wirklich frei zugänglich ist, weil es AT&T Code enthält.

2 Das FreeBSD Projekt

Das FreeBSD Projekt entstand, wie schon erwähnt als sich das 386BSD Projekt spaltete, weil die Interessen der einezelnen Beteiligten zu divergent waren. Die speziellen Ideen des FreeBSD Projektes waren:

- sich auf die i386 Hardware (als billigste und weitverbreiteteste) zu konzentrieren
- das System für den Benutzer möglichst komfortabel zu gestalten (einfache Installation, einfache Administration,...)

Bei der Organisation wurde u.a. auf Erfahrungen von der Universität Berkely zurückgegriffen. FreeBSD hat ein **Core Team**, welches für grundsätzliche Entscheidungen zuständig ist und in gewisser Weise die Verantwortung für das ganze Projekt trägt. Dann gibt es eine größere Gruppe von **Committern**. Das sind Leute die Schreibrechte im CVS Repository von FreeBSD haben. Weiterhin gibt es **Entwickler** oder **Contributoren** die selbst keine Schreibrechte im CVS Repository haben und ihren Code über einen Committer in FreeBSD einfließen lassen. Jeder der eine Weile als Contributor mitgearbeitet hat, kann selbst Committer werden. Während seiner ersten Zeit erhält er einen älteren Committer als Mentor. Das Core Team wird aller vier Jahre Jahre von den Committern gewählt.

Für FreeBSD Entwickler und speziell Committer gibt es nicht nur einen Style Guide für den Code den sie schreiben, sondern vor allem einen **Committers Guide**, wo die generellen ”Spielregeln” für die Teilnahme an diesem Projekt festgelegt sind. Dieser umfaßt auch Kapitel über den generellen Umgang miteinander und speziell das Mailen auf FreeBSD internen Mailinglisten. Dieser Committers Guide ist sicher nicht unerheblich dafür verantworltich, daß dieses Projekt trotz der relativen Größe und großen Anzahl an beteiligten Personen relativ harmonisch weiterentwickelt wird.

Im CVS Repository von FreeBSD wird mit verschiedenen Zweigen (branches) gearbeitet. Jedes Release entspricht einem Branch. Es sind immer zwei Branches ”in Entwicklung”. Ein -STABLE Branch, wo vor allem Bug Fixes und Security Patches eingespielt werden. Neue Features werden hier nur sehr vorsichtig aufgenommen. Und ein

-CURRENT Branch, wo vor allem an neuen Features gearbeitet wird. Neue Features müssen immer nach -CURRENT committet werden und können erst nach einer Testphase in -STABLE übernommen werden. Dadurch wird sichergestellt, daß der -STABLE Branch seinen Namen auch zu recht trägt.

Im Vergleich zu anderen OpenSource Betriebsystemen muß zu FreeBSD noch gesagt werden, daß es sich um ein komplettes Betriebssystem mit einer eigenen C-Library und einer Menge Tools, und nicht bloß um einen Kernel handelt.

Ein weitere Punkt der teilweise sehr leidenschaftlich und auch emotionsgeladen diskutiert wird, ist die Lizenz unter der FreeBSD steht, die Berkely License. Diese Lizenz ist deutlich freier als die GPL, da sie es u.A. erlaubt, den Code zu verändern und in binärer Form zu verkaufen, wobei nur der Hinweis auf die Urherberrechte der Autoren und der University of California at Berkely enthalten sein muß.

3 FreeBSD Specials

Das Ziel ein möglichst einfach zu verwaltendes Betriebssystem zu schaffen, hat einige sehr Interessante Eigenschaften in FreeBSD hervorgabracht, die zumindest kurz erwähnt werden sollten. Diese sind vor allem der *make* Mechanismus, der FreeBSD komplett durchzieht und das Modell für Third Party Software, der *ports* und *pkg** Mechanismus.

Wer FreeBSD mit den kompletten Sourcen installiert, erhält einen Sourcetree, in dem es möglich ist nur mit dem Befehl *make world* das komplette Betriebssystem neu zu compilieren und zu installieren. Dabei ist können in der Datei `/etc/make.conf` globale Parameter (z.Bsp. Compileroptionen) gesetzt werden. Diese werden bei jedem *make* Aufruf innerhalb von FreeBSD verwendet. Dazu gehört einmal das Compilieren des Betriebsystems *make world*, das Compilieren des Kernels *make kernel* und auch das Compilieren von Third Party Software aus den Ports.

Die *ports* sind ein Verzeichnisbaum für Software, die nicht zu FreeBSD selbst gehört, wie z.Bsp. XFree oder der KDE. Es gibt für jede Software ein kleines Makefile eine Beschreibung des Paketes und u.U. patches. Wenn der Administrator in einem dieser Verzeichnisse *make* oder *make install* sagt, dann überprüft make ob die Sourcen des Paketes schon auf dem System liegen, wenn nicht werden sie per ftp heruntergeladen, das Paket wird ausgepackt, configuriert compiliert und installiert. Dabei werden Abhängigkeiten aufgelöst und abhängige Pakete ebenfalls installiert. Bei einem FreeBSD Rechner mit Internetanschluß reduziert sich das Installieren des KDE z.Bsp. auf den Befehl *make install* und das warten, bis der Rechner fertig ist. (Was bei älteren Rechnern auch sehr lange dauern kann.) Neben dem *install* Target gibt es in den Ports noch weitere Targets. Eines ist *package*. Mit dem Befehl *make package* wird von der etnsprechenden Software ein binär Paket erzeugt, welches sich mit dem FreeBSD Package Mechanismus *pkg_(add|delete|info|...)* installieren läst. Beim Auflösen von Abhängigkeiten ist es egal, ob ein anderes Paket über die ports oder mit dem pkg* Mechanismus installiert wurde, es wird in jedem Fall gefunden. Die beiden Mechnismen sind absolut kompatibel zueinander.