Betriebssysteme

Prof. Neeraj Suri, Dr. Stefan Winter

Wintersemester 2018-2019

Inhaltsverzeichnis

1 Geschichte

Am Anfang gab es noch keine Betriebssysteme, alle Programme liefen direkt auf der Hardware (*Bare Metal*). Da jedes Programm in irgendeiner Weise Daten verarbeiten muss, musste man für jedes Programm gewisse Eingaben zugänglich machen, und gewisse Ausgaben in einer für dem Benutzer nutzbaren Art ausgeben. Also musste man Code schreiben, der eigentlich garnichts mit dem Programm zu tun hat, sondern nur dazu dient, diese Eingaben und Ausgaben zu ermöglichen.

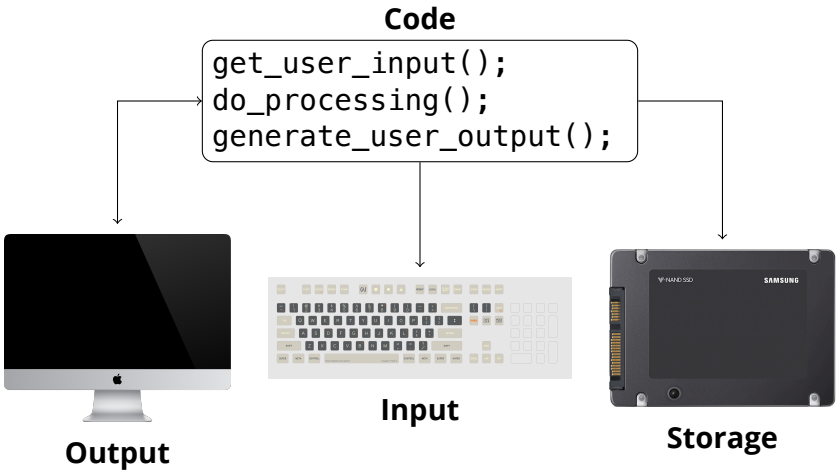


Abbildung 1: Programmierung auf Hardware ohne Abstraktion

1.1 Library

Irgendwann erkannte man dann, dass man sehr oft gewisse Funktionalität neu implementieren musste, weil unterschiedliche Programme ähnliche Eingaben erwarteten oder ähnliche Ausgaben erzeugten, und man diesen Code dafür widerholte. Eine Idee hier ist, den Code, den man häufig verwendet, in eine *Library* packt. Damit können diese Routinen wiederverdendet werden. Außerdem können Programme sich so mehr auf ihre eigentlichen Aufgaben konzentrieren.

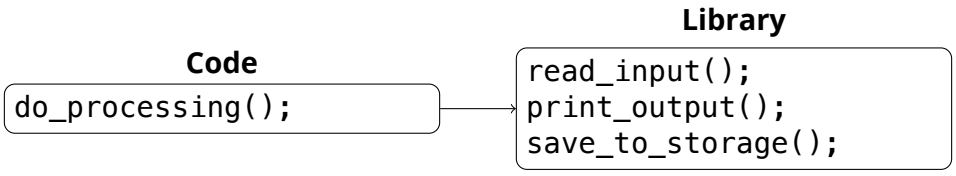


Abbildung 2: Ausbau von Funktionen in eine Library

Der Nachteil von diesem Ansatz war, dass diese Libraries immer wieder umge-

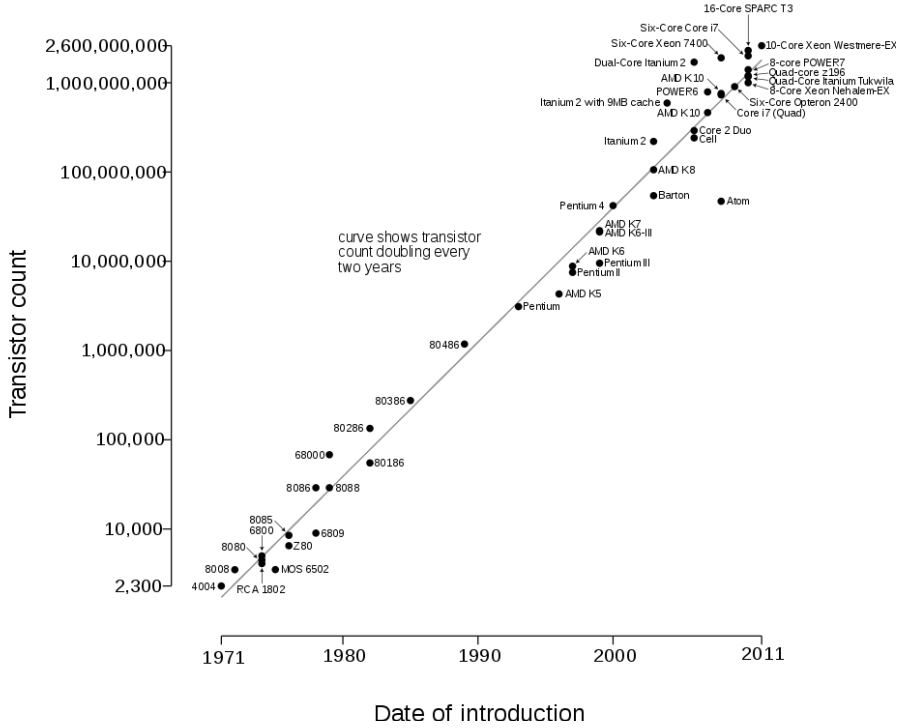


Abbildung 3: Hardwarekomplexitätswachstum seit 1971.

schrieben mussten, wenn ein anderer Computer verwendet wurde.

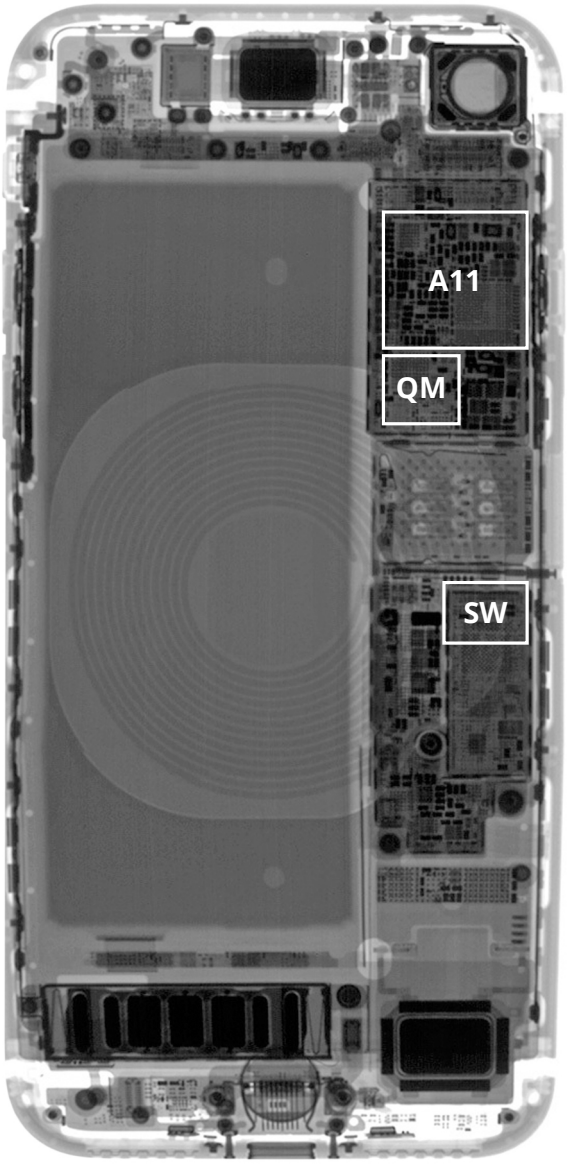
1.2 Exponentielles Wachstum von Hardwarekomplexität

Es ergab sich also die Schwierigkeit, dass die Hardware sich so rasant entwickelt. Dies ist in Abbildung 3 graphisch aufgezeigt, wobei man anmerken muss, dass die Skala logarithmisch ist. Das scheinbar lineare Wachstum der Transistoranzahl ist also tatsächlich ein exponentielles. Das nennt man auch Moore's law, nach dem Begründer von Intel.

1.3 Heute

Heutzutage sind Betriebssysteme *überall*. Von Fernsehehern zu Autos, Servern, Handys bis hin zu Operationssälen. Interessanterweise sind teilweise mehrere Betriebssysteme auf einem Gerät zu finden. Abbildungen 4 und 5 zeigen, dass sogar in den Geräten, die wir tagtäglich verwenden, oft mehr steckt, als wir denken.

Die Betriebssysteme, die wir benutzen, haben oft sehr viele Features eingebaut.



Apple A11 SoC
Betriebssystem: iOS (Darwin), UNIX-Derivat, POSIX kompatibel.

Apple Secure Enclave
Im A11-Chip eingebaut
Betriebssystem: Basiert auf dem L4 Mikrokernel

Qualcomm Snapdragon X16
LTE Modem
Betriebssystem: Unbekanntes RTOS, eventuell REX OS

Skyworks SkyOne SKY78140
CDMA Modem
Betriebssystem: Unbekannt

Abbildung 4: Betriebssyteme auf Konsumergeräten am Beispiel iPhone

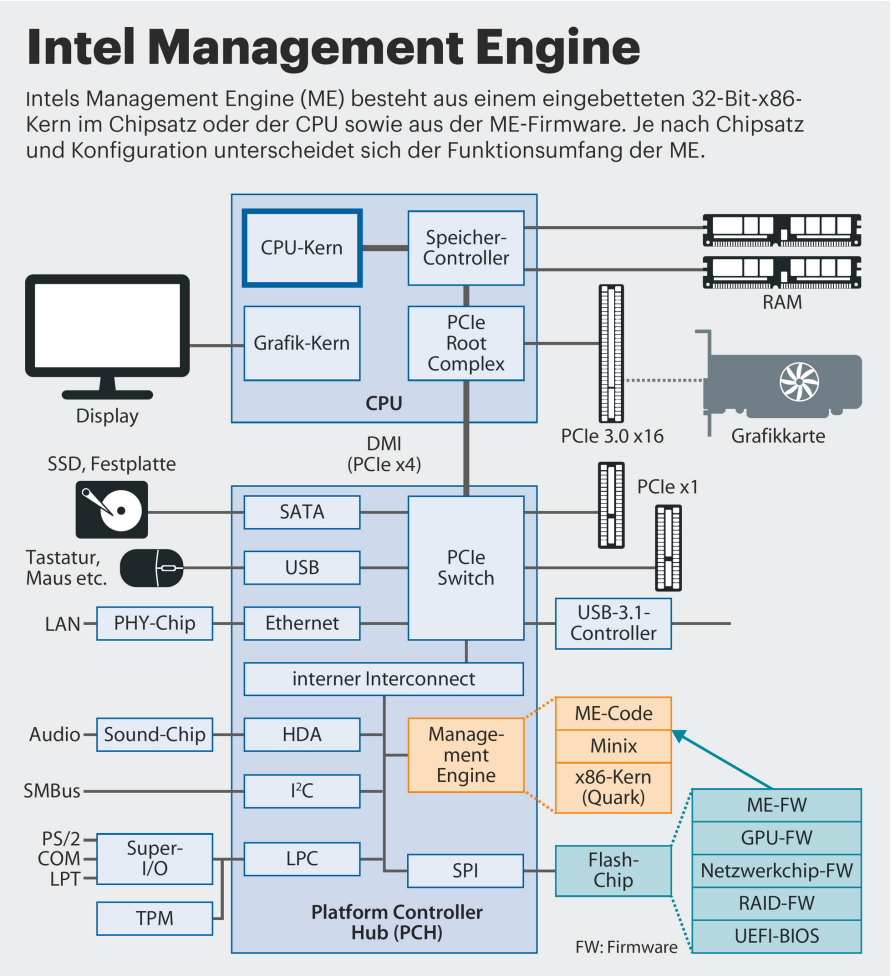


Abbildung 5: Intel Management Engine. Quelle: Heise.de

2 Definition

Wie definiert man eigentlich, was ein Betriebssystem ist, und was nicht? Dazu gibt es nicht nur eine, sondern gleich mehrere Definitionen.

2.1 Hardwareabstraktion

Man kann ein Betriebssystem als eine Hardwareabstraktionsschicht beschreiben. Das Betriebssytem ist als dafür zuständig, eine Ausführungsumgebung zu erstellen für die Programme, die darauf laufen sollen. Dazu muss das Betriebssystem den Programmen eine Abstraktionsebene bieten, damit diese nicht direkt mit der Hardware interagieren müssen. Außerdem muss das Betriebssyste die Resourcen verwalten und (idealerweise fair) zwischen den Programmen teilen.

Wenn man sich jetzt mal anschaut, wie viel Code eigentlich hinter der Ausführung eines kleinen, simplen Programms steckt, dann merkt man, wie viel eigentlich hinter den Kulissen passieren muss, damit die Programme das tun, was wir von ihnen erwarten. In Abbildung 6 ist zu sehen, welche Ebenen eigentlich unter einen Programm liegen, und wie viel Code diese Beinhalten.

Ebene	**Beispiel**	**Zeilen**
Java Bytecode		~1 000
Java Runtime	`openjdk11`	8 047 913
Support Libraries	`glib`	460 641
	`libcxx`	441 343
System Libraries	`glibc`	1 384 092
Kernel	`darwin`	1 070 917
	`openbsd`	2 235 267
	`netbsd`	5 930 334
	`linux`	17 120 205
	`windows`	~65 000 000
Hardware		

Abbildung 6: Hardwareabstraktionsebenen und Codegröße

Anmerkung
Ich habe mir mal die Freiheit genommen, die Zeilen Sourcecode für die meisten Projekte hier durchzuzählen. Dazu habe ich das Programm `cloc` benutzt (auf macOS mit `brew install cloc` installierbar). Die `libcxx` ist die C++ Standard Library, die vom Clang Compiler des LLVM Projekt genutzt wird (wird verwendet von iOS, macOS, Linux, usw.). Der macOS Kernel, `darwin`, ist Quelloffen, deswegen habe ich den auch mitgenommen. Allerdings muss man dazu sagen, dass dieser modular aufgebaut ist, und hier keine Module mitgenommen wurden, deswegen sind die Resultate nicht direkt verlgeichbar. Bei den BSD habe ich jeweils den `/sys/`-Tree des Sourcecodes ausgewertet.

Außerdem musste man ja.