Linux: Wat is Open Source?

D. Leeuw

13 april 2025
v.1.1.2

Over dit Document

Inhoudsopgave

Hoofdstuk 1

Waarom Open Source?

Wat Linux en het GNU project bijzonder maken is het feit dat alle code open source is. Je mag er mee doen en laten wat je wilt, je mag het aanpassen, je kan het inzien en je mag het natuurlijk gewoon gebruiken. Voor het merendeel is de software nog gratis ook. Dat is natuurlijk bijzonder in een wereld die draait om commercie. Daarom willen hier iets dieper duiken in de wereld van open source.

De eerst versies van Unix zoals geschreven door Ken Thompson, Dennis Ritchie en de overige leden van het team bij Bell Labs was geschreven is de assembly language. Assembly language is een taal die heel dicht ligt bij wat computers snappen en daarmee altijd hardware afhankelijk is. In de tijd dat Unix werd ontwikkeld geloofde men dat je assembly language nodig had om een besturingssysteem snel genoeg te laten zijn. Dennis Ritchie nam de taal B, ontwikkeld door Ken Thompson, maakte verbeteringen en kwam met C in 1972. In 1973 kwam Unix versie 4 uit die voor een groot deel herschreven was in C en daarmee aantoonde dat een hogere programmeertaal gebruikt kon worden om besturingssystemen in te schrijven die snelgenoeg waren, maar belangrijker nog omdat er een hogere programmeertaal werd gebruikt was Unix opeens overdraagbaar naar andere hardware en dat had voor de ontwikkeling van software grote voordelen. Software kon opeens geschreven worden op het ene systeem en gebruikt worden op een totaal ander systeem.

Voor het programmeren in C heb je een C-compiler, een linker en een C-Library nodig. Library is het Engelse woord voor bibliotheek. Een C-bibliotheek bevat een aantal standaardfuncties die je kan gebruiken in een programma. Een heel simpel programma als Hello World ziet er in C zo uit:

```
#include <stdio.h>

int main() {
        printf("Hello, World!");
```

```
        return 0;
}
```

De printf functie die de woorden "Hello World!" op het scherm laat zien is zo'n standaard functie uit de C-Library. De C-bibliotheek bevat een aantal standaardfuncties die je kan gebruiken in een programma, zo hoeft niet elke programmeur de printf functie te programmeren. Functies in een library zijn dus kleine stukjes code die je met elkaar delen kan zodat iedereen in zijn programma deze functies gebruiken kan mits de library op het systeem aanwezig is.

De compiler is verantwoordelijk voor het omzetten van de C-code in machinetaal. Machinetaal is binair, daar computers werken met 1 en 0 en niets anders. Het proces om C-code om te zetten in binaire code heet dan ook compileren. De compiler vertaalt alles wat er geschreven is in C in 1-en en 0-en zodat de computer ermee kan werken. Maar omdat je ook functies gebruikt uit de C-library moet ook daar nog iets mee gebeuren, daarvoor zorgt de linker. De linker zorgt ervoor dat de functie uit de C-library gelinkt wordt aan het programma dat je geschreven hebt. Een compiler is niet alleen hardware afhankelijk, maar ook operating systeem afhankelijk. Een compiler voor Mac OS X maakt van C machinetaal voor Mac OS X en een C-compiler voor Windows maakt machinetaal voor Windows.

Er zijn twee manieren waarop de linker ervoor kan zorgen dat bijvoorbeeld de printf-functie gelinked kan worden met je programma. Het kan statisch en dynamisch. Statisch betekent dat een kopie van de functie toegevoegd wordt aan je programma. Bij dynamisch linken betekent het dat er in je programma een verwijzing komt te staan naar de binaire printf-functie in die specifieke binaire C-library. Je C-programma wordt zo afhankelijk van deze specifieke versie van de C-library die aanwezig is op je systeem.

Het voordeel van statisch linken is dat het programma onafhankelijk is van de C-library, het nadeel is dat het binaire-programma vele malen groter wordt omdat dat alle code uit de C-library (of andere bibliotheken) ook aan het programma wordt toegevoegd. Bij dynamisch linken is dit juist omgekeerd. Het programma blijft kleiner, laadt daardoor sneller van disk en neemt minder geheugen ruimte in, maar dat gaat tenkoste van de portabiliteit, kortom het kan alleen nog gebruikt worden op systemen die exact dezelfde versies van de libraries geïnstaleerd heeft. Dat laatste is geen probleem zolang je het programma gebruikt op systemen die dezelfde libraries hebben als jij, zoals het geval is bij mensen die dezelfde distributie gebruiken als jij. Ook als er een kleine wijziging gemaakt wordt in de printf-functie die geen invloed heeft op de binaire syntax van de printf-functie dan kan je programma gelijk gebruik maken van deze verbetering doordat je alleen de

C-library update. Je hoeft je programma dan niet opnieuw te compileren. Er zijn dus vele voordelen aan dynamisch linken en daarmee is het dan ook de meest gebruikte methode op Linux systemen.

Dat dynamisch linken klinkt allemaal vreselijk complex en dat is het ook. Er zijn momenten waarop er zoveel wijzgingen zijn in bijvoorbeeld een nieuwe C-library dat jij je programma opnieuw moet compileren om het te laten werken met die nieuwe C-libary, maar kleine wijzigingen in de C-library kunnen ervoor zorgen dat dat niet hoeft en dan blijft je programma gewoon werken. Dat verschil heeft te maken met de API (Application Programming Interface) en ABI (Application Binary Interface). De API van een functie is de syntax van de functie, als deze verandert dan moet je je programma opnieuw compileren en als het tegenzit moet je zelfs je code aanpassen. Als echter de API blijft zoals hij is en er zijn alleen kleine wijzigen zodat de ABI niet veranderd, dan heeft je programma geen last van de wijziging.

Programma's die werken op je telefoon, je Windows systeem of op Mac OS X worden je vaak aangeboden als binary, kortom ze zijn al door iemand gecompiled. Deze applicaties kan je direct gebruiken, maar alleen op het systeem waarvoor ze gecompiled zijn. Je kan een Windows .exe niet gebruiken op een Mac OS X systeem.

Als je (ook) de beschikking hebt over de broncode dan kan je die code ook compilen op je eigen computer en zorgen dat die ook werkt op jou systeem. Je bent dan niet meer afhankelijk van een leverancier die jou systeem moet ondersteunen. Soms moet je wel wat aanpassingen maken om het geheel goed te laten werken. De grafische interface van Windows is heel anders dan die van Mac OS X, en daar zit dan ook een heel andere library onder. Dus als je een Windows applicatie op een Mac wil compileren zul je wel wat programmeer werk moeten doen. Maar als een applicatie is geschreven op een Debian systeem dan kan deze meestal zonder enige wijziging gecompileerd worden op een CentOS systeem.

En daar zit de kracht van open source. Met het delen van de broncode wordt de reikwijdte van die software groter. Zoals gezegd moet je soms wel wijzigen maken om het te laten werken en dus is het bijna een eis dat je de software ook mag aanpassen en die eisen zijn in open source licenties vastgelegd.

Index