Waarom gratis Linux toch te duur kan zijn?

Veel embedded producten staan onder een geweldige prijsdruk. Het moet continu beter en goedkoper. Linux is dan verleidelijk: het heeft geen dure licenties. Maar daalt de kostprijs daarmee? Of zitten de kosten elders?
Linux is bezig met een sterke opmars. Het oorspronkelijke hobbyproduct, dat de servermarkt deels veroverde en in opmars is op de desktop, heeft ook goede kansen bij de onbekendere delen van de computerindustrie. Zowel op het mainframe; maar ook in embedded systemen is Linux in opkomst. Steeds meer moderne elektronicaproducten bevatten al 'gratis' Linux.
Dat Linux gratis is, is vooral voor studenten een belangrijk argument, hun avond(t)uren kosten niets. Maar echte producten moeten ontwikkeld worden door hoog geschoold, duur, technisch personeel. Dat kost geld. Daarom de vraag: Hoe gratis is Linux?

Inhoud

Wat kost een embedded systeem?

Er zijn vele vormen van embedded software, met namen als 'in product software', 'dedicated systeem' en 'realtime'. Hoewel er zeker onderscheid gemaakt kan worden, zal ik het eenvoudig houden: een product dat om te functioneren een CPU en programmatuur bevat, noem ik een embedded systeem. Meestal zal zo'n product als geheel verkocht worden. Dit maakt embedded software anders dan bij 'normale computers', waar hard- en software apart verhandelbaar zijn. En waar 'normale' systemen 'beheerd' worden door een uitgebreide groep mensen –van helpdeskmedewerker tot ICT-manager– heeft embedded software geen 'gebruikers' of 'beheerders'. Zeker niet op softwareniveau, hooguit op systeemniveau. Dat is een fundamenteel verschil.

Er zijn geen beheerskosten!

Het hierboven beschreven verschil tussen embedded en 'normale' computersystemen heeft veel effect op de kostenberekening. Met grote regelmaat verschijnen er –al dan niet gesponsorde– overzichten of Linux of Windows goedkoper zou zijn. We gaan hier niet in op de juistheid van dergelijke berichten, maar we willen wel benadrukken dat het hier om 'beheerde computers' gaat. Hier vormen de beheerskosten vaak het merendeel van de totale kosten en software-ontwikkelkosten bestaan hier niet.
Een kostenberekening voor embedded producten zal er daarom geheel anders uitzien. Elk embedded product is uniek en de kostenberekening is altijd anders. Toch kunnen we vier

hoofdcategorieën qua kosten onderscheiden. Ten eerste de ontwikkelkosten; deze zijn typisch eenmalig. Daarna komen de productie-, fabricage- of reproductiekosten: de prijs die betaald moet worden per gemaakt (of verkocht) product. Voor software zijn dat voornamelijk licentie- en installatiekosten De derde categorie zijn de migratie kosten; denk aan het opleiden van ontwikkelaars. Ook het tijdelijke productiviteitsverlies in de omschakelperiode en nieuwe tools en methoden vallen hieronder.
De vierde categorie is vaak onderschat: de 'kruisposten'. Dit zijn indirecte kosten, die het gevolg zijn van keuzes –vaak lokale besparingen– op een andere plaats. In de volgende paragraaf gaan we daar dieper op in.
Kosten voor marketing, PR, sales, e.d. zijn voor dit artikel niet relevant en laten we buiten beschouwing.

Vergeet de kruisposten niet!

Een embedded systeem bestaat vaak uit complexe deelsystemen, die samen één geheel moeten vormen. Elk deelsysteem wordt zoveel mogelijk onafhankelijk ontwikkeld. Zo is de hardware 'onafhankelijk' van de software. Projectmatig is dat verstanding. Maar levert dat altijd de laagste totaal kosten op?
Keuzes in het éne deel beperken de oplossingsruimte van andere delen. En kunnen zo extra kosten introduceren.
Zo kunnen de keuze voor een besturingsysteem en de hardwarekeuze niet los van elkaar gezien worden.
Ooit duurde een presentatie over hun kansen met Windows-CE slechts vijf minuten; omdat de keuze voor een PowerPC en de keuze voor dit realtime besturingsysteem niet samen gaan. Microsoft ondersteunde na versie-3.0 die combinatie niet meer. Al is het jammer van de presentatie, door dit kruisverband vroegtijdig te ontdekken zijn kosten bespaard.

Meestal zijn deze kruisrelaties echter niet zo duidelijk, al heeft de besturingsysteem-keuze altijd veel effect. Het heeft effect op zowel de andere software als op de hardware. Soms positief, soms negatief. Zo kan Linux het ontwikkelen van een embedded webserver eenvoudiger maken, maar ook meer geheugen vragen. Vaak denkt men wel aan de verhoogde productiekosten, maar vergeet men de besparing op ontwikkelkosten. Terwijl het extra geheugen soms geen effect heeft op de kosten. Er zijn voorbeelden waar de besparing op geheugengebruik als enige effect had dat een (nog) groter deel van het beschikbare geheugen niet gebruikt werd! De grootte van moderne geheugenchips maakt dit traditionele doel soms overbodig.
In het algemeen zijn de kosten van dergelijke relaties lastig te ontdekken omdat een gedetailleerd overzicht nodig is van de gehele ontwikkeling. Van zowel hardware als software. Maar bovendien vergt het inzicht in ontwikkel- en fabricagekosten. En dus is het nodig om het aantal te fabriceren producten te weten om de maximale besparing te kunnen bepalen.
In praktijk is alles te complex; daarom gebruikt men vuistregels die in praktijk effectief blijken te zijn.
De vraag is echter of die vuistregels –en dus de rekenwijze– nog geldig zijn na een overgang op Linux.

Risico's en besparingen

Een andere kostenpost betreft 'tegenvallers'. Elke project heeft onvoorziene kosten; in elk plan is daar een budget voor gereserveerd. Oparationeel is dat erg praktisch, ook al worden de werkelijke kosten hiermee versluierd. Daarom gebruiken we hier een andere, wat meer academische aanpak. Vrijwel elke tegenvaller is (achteraf) te herleiden naar een onzekerheid. En dus naar kosten, met een (vaak kleine) kans dat die post opgenomen moet worden. Door de maximale kosten te vermenigvuldigen met de kans dat ze optreden kunnen ze stuk voor stuk in kaart gebracht worden. Waardoor ze op te nemen zijn in één van de genoemde categorieën.
Op dezelfde manier kan omgegaan worden met besparingskansen: De maximale besparing maal de kans op die besparing wordt als negatieve kostenpost opgenomen.

Op deze manier kunnen besparingen in de ene categorie verrekend worden (extra) kosten in een andere.

- **Een snelle besparing**

Ter illustratie: een automatisch meetnetwerk met zo'n 55 nodes. Het project dat dit opleverde, gebruikt (embedded) Linux op een industriële PC met 'compactFlash' als harddisk. Uit berekeningen blijkt dat een 64Mbyte 'disk' nodig was. Omdat compactFlash ook gebruikt wordt in bijvoorbeeld digitale fototoestellen, is een bezoekje aan een winkel voldoende om te leren dat (destijds) een kaart van 128Mbyte zo'n €15 duurder zou zijn. In totaal dus slechts €815 extra 'fabricagekosten'.
Het voordeel van deze extra diskruimte is dat men een standaard distributie kan gebruiken, inclusief een gemakkelijke installatie. Dat is niet echt nodig, maar de kans is groot dat er weken of maanden ontwikkeltijd bespaard kunnen worden. Bijvoorbeeld omdat men niet goed weet hoe Linux te verkleinen.
Wat zou u doen? Besparen op de hardware? Of toch maar niet?

Weinig licenties zijn ook goedkoop!

In bovenstaand voorbeeld werden de ontwikkelkosten verdeeld over een klein aantal te fabriceren producten. In de regel ligt dat bij consumentenproducten geheeld anders. De kosten van extra hardware en de (software) licentiekosten zullen dan zwaar wegen. Maar lang niet altijd zullen de ontwikkelkosten per product te verwaarlozen zijn!
Te vaak wordt een degelijke berekening niet goed gedaan. Men staart zich dan blind op het 'gratis' Linux, terwijl een alternatief als Windows-CE maar zo'n drie dollar per licentie kost. U kunt zelf uitrekenen welke aantallen u nodig heeft om een poort van Windows naar Linux, qua kosten, te rechtvaardigen.
Natuurlijk kunnen 'kosten' niet het enige argument zijn. Om te kiezen voor Linux kunnen ook doorlooptijd, 'time-to-market' en kwaliteit een rol spelen.
Kijken we puur naar de kosten, dan moeten we constateren dat de ontwikkelkosten vaak een aanzienlijk deel van de kostprijs bepalen. En dat het soms goedkoper is om extra hardware te gebruiken om zo de software (veel) sneller te kunnen ontwikkelen.

De kosten van embedded Linux

We spreken van 'embedded Linux' als Linux gebruikt wordt in een embedded systeem. De functionaliteit van het product verandert daarmee niet. Hooguit kan Linux helpen bepaalde functionaliteit gemakkelijker te ontwikkelen. Het al dan niet gebruiken van Linux is een ontwerpkeuze. Zowel techniek, beleid en kosten spelen hierin een rol. Wat de reden ook is, de keuze voor Linux heeft effect op de kosten. Kiezen voor Linux kan betekenen dat de licentiekosten, als onderdeel van de reproductiekosten, verminderen. Maar kiezen voor Linux heeft ook effect op de ontwikkelkosten! En heel vaak ook op de kruisposten.

Verborgen kosten

Dat de keuze voor Linux vaak een besparing oplevert op gebied van licentiekosten ligt voor de hand. De vraag is of deze 'negatieve kosten' opwegen tegen de kosten op ander vlakken. De kosten voor opleiding en nieuwe tools zijn redelijk voor de handliggend. De grootste tegenvallers ontstaan vaak door eerder genoemde kruisposten. En omgekeerd worden potentiële besparingen niet gehaald, omdat die niet in de kostenberekening meegenomen zijn.

Praktijkvoorbeelden

In de praktijk vallen de kosten van embedded Linux nogal eens tegen. Vooral als er uitgegaan wordt van verouderde ervaring –zoals vuistregels gebaseerd op traditionele systemen. Dan zijn tegenvallers niet te vermijden. Bovendien worden potentiële besparingen niet bereikt.

Zo zijn er talloze voorbeelden van achteraf gezien makkelijk te voorkomen extra kosten.

- **Dubbele ontwikkeling**

Mensen hebben bij een overschakeling naar een ander systeem vaak de neiging om oude gewoontes vast te houden: Door op het nieuwe systeem, het oude te emuleren. Beginnend Linux programmeurs kunnen gewend zijn aan het DOS commando 'dir'. In plaats van de unix tegenhanger 'ls' te gebruiken, wordt het soms 'opgelost' door een scriptje te introduceren, genaamd 'dir' dat het ls-commando uitvoert. En hoewel men daarmee sneller kan werken en zeker minder geërgerd is, leert men het nieuwe systeem niet kennen.
De problemen ontstaan pas later, als men nieuwe programma's maakt die deze 'compatibiliteit' gebruiken. Zo kan men bijvoorbeeld een gesorteerd overzicht maken door 'dir' aan te roepen, het resultaat tijdelijk op te slaan, die file te sorteren en dan te printen naar het scherm. Een gebruikelijke constructie op een DOS systeem. Maar op Linux kan het veel efficiënter; het ls-commando heeft namelijk diverse opties om te sorteren.
Bij de overgang naar embedded Linux gebeurt hetzelfde. Pas nadat één of twee producten gemaakt zijn, blijkt het Linux gevoel goed in de vingers te zitten. Er zijn maar weinig bedrijven die het aandurven om dan met een nieuw referentiemodel te komen en niet verder te bouwen op die minder efficiënte 'tussenproducten'. Een enkeling doet het structureel beter: ze plannen een overgang in 2 stappen. Ze weten dat ze beide nodig hebben en zien de eerste poging als 'leerfase'. De investering hiervan is tijdelijk en wordt in fase 2 dubbel en dwars terugverdiend.

- **Ongelukkige keuze maakt Linux duur**

In een telefoonsysteem was gekozen om de codec-chip direct op de CPU aan te sluiten en niet op een FPGA –wat bij een vergelijkbaar ontwerp wel gedaan was. Voor het gebruikte RTOS was dat geen probleem. Bij de overgang naar Linux zorgde deze ene 'ongelukkige' keuze voor een hard-realtime eis van 200 microseconde! Terwijl er anders slechts een soft-realtime eis van 50 milliseconde zou gelden.
Embedded Linux heeft een drempel van rond de 20 milliseconde; daarboven is er vrijwel nooit een probleem. Terwijl responstijden onder de 20 milliseconde aandacht nodig hebben. Sneller kan, zeker in interrupts. Maar in dit geval met meer dan duizend hard-realtime interrupts per seconde was dat moeilijk.
Deze ene hardware keuze had daarom een dramatisch –negatief– effect op de Linux kosten. Minimaal was een extra RT uitbreiding nodig, met de bijbehorende kosten. Ook werd het project veel complexer en daardoor duurder.

- **Informeel gebruik van een API**

Bij traditionele RTOS'en wordt soms gebruik gemaakt van wat een 'informele API' genoemd zou kunnen worden. Zo kan een programmeur bijvoorbeeld de tijd te weten komen door een register uit te lezen. Maar in een Linux applicatie is dat onmogelijk! Er is wel een API om de tijd op te vragen; maar het aanroepen van die functie kost veel meer tijd.
Daarom zal naast het zoeken naar plekken waar zo'n informele API gebruik wordt, het ontwerp waarschijnlijk moeten veranderen om dat te compenseren. Deze twee acties kunnen veel tijd kosten. Soms is het zelfs goedkoper om delen geheel opnieuw, voor Linux, te ontwerpen.
Vrijwel altijd zijn er van dit soort tegenvallers. Ook een standaard als 'POSIX' blijkt soms slechts op papier te bestaan. En minder compatibiliteit te geven dan gehoopt.

- **Verkeerde aannames en onjuist testen**

Veel OpenSource producten zijn zowel beschikbaar voor Windows als Linux. Maar het is onjuist om aan te nemen dat het dezelfde producten zijn. Zo is het algemeen bekend dat het 'pre-fork executiemodel', dat apache oorspronkelijk bedacht voor Unix systemen en daar een geweldig performance gaf, op Windows nauwelijks werkt. Apache heeft dit inmiddels opgelost met een flexibel, modulair executiemodel, 'MPM' genaamd. Ook op Windows wordt nu een goede performance behaald. Maar wat daar zorgt voor de nodige snelheid, werkt op Linux weer niet. Een

verkeerde keuze van de 'MPM' voor het te testen platform maakt de test waardeloos. Functioneel werkt het, maar de respons(tijd) is volkomen anders.
Dit probleem trad, meer verhuld, op in een embedded Linux product, dat gebruik maakte van OpenSource mpeg-bibliotheken. Omdat men niet bekend was met mpeg, werden uitgebreide feasibility-testen uitgevoerd. Voor het gemak deed men dat met de windowsversie, die eenvoudiger te downloaden en te gebruiken was. Later bleek dat de gebruikte configuratie niet ondersteund werd op Linux.
Uiteindelijk is er wel een werkende embedded Linux versie opgeleverd, maar de testen hebben daar nauwelijks aan bijgedragen.

- **Het wiel opnieuw ontdekken**

Er is een aantal OpenSource producten die in vrijwel elk embedded Linux systeem te vinden is. Vaak is het specifiek ontwikkeld voor embedded systemen, maar verder vrij onbekend. In veel nieuwe embedded Linux projecten volgt men dan ook een standaard patroon: men gaat hard opzoek naar allerlei mogelijke oplossingen, probeert de beste te selecteren en komt vervolgens uit op 'de Busybox'. Dit is zo'n standaard oplossing.
In een onlangs opgestart project deed zich een variant hierop voor. Hier werd uitgegaan van standaard oplossingen (er was embedded Linux ervaring). Waarna een manager aanstuurde op een overleg met een andere medewerker; die een geweldig product kende ... inderdaad, de Busybox.

Conclusie

Dat Linux goed gebruikt kan worden voor embedded systemen staat niet ter discussie. Wel is aan de hand van een aantal cases aangetoond dat 'gratis' geen overtuigend argument is om voor Linux te kiezen. De kostprijs wordt immers bepaald door meer dan alleen de hardware- en licentiekosten. De ontwikkelkosten zijn niet te verwaarlozen. Bovendien zullen er 'tegenvallers' zijn; vooral als men uitgaat van 'verouderde' vuistregels. Niet alleen zijn er daardoor onnodige kosten, ook worden potentiële besparingen niet benut.
Natuurlijk zijn er andere argumenten dan 'kosten' om voor Linux te kiezen. Maar zelfs als we uitsluitend naar de kosten kijken, zijn er goede voorbeelden. Maar het wordt niet automatisch goedkoper, alleen omdat u voor Linux kiest! Lees daarom het artikel *"Ik wil weten hoe Linux snel en goedkoop kan"*.

ALbert Mietus