Embedded software: strategic issues

Embedded software: strategic issues

Brent Frère,

LiLux

Linux Days 2002

Some definitions

Some definitions

- Free Software

Free software does not means 'gratis software' but software respecting the following principles:

Some definitions

- Free Software
  - Freedom 0: Freedom of execution

Whenever country you are in.
For whatever environment: business, private, educational purpose. Free Software doesn't mean 'non commercial'. Examples of commercial free software: Linux distributors (Suse, RedHat, Mandrake, Caldera, Slackware...), MySQL, IBM...
Whenever the purpose of the execution is.
Whatever the power of the executing host is.
Even if the software is used concurrently by numerous users.

Some definitions

- Free Software
  - Freedom 0: Freedom of execution
  - Freedom 1: Freedom to adapt, modify, study the soft

Implies the availability of the sources.
Allows learning process. Learning computer skills on a closed systems is not learning: it's training how to use commercial product.
Allows adaptations for a specific use.
Allows fast correction process.
Prevents spy code, secret back-doors.
Enhance security by white-box testings, and fast security fix availability.

Some definitions

- Free Software
  - Freedom 0: Freedom of execution
  - Freedom 1: Freedom to adapt, modify, study the soft
  - Freedom 2: Freedom to distribute copies

For free or against a fee.
Free software doesn't mean 'gratis'.
The copies must be distributed under the same license terms, in order to guarantee the software to remain free.

Some definitions

- Free Software
  - Freedom 0: Freedom of execution
  - Freedom 1: Freedom to adapt, modify, study the soft
  - Freedom 2: Freedom to distribute copies
  - Freedom 3: Freedom to enhance the soft

You can modify the software, enhance, add functionalities and distribute the new obtained version under the same license terms.

Some definitions

- Free Software
- Free-Software foundation

Created by Mr Richard Stallman in 1985, originator of the 'Free Software' concepts. This organization created the 'CopyLeft', GPL and L-GPL licenses, and has developed the GNU project.

GNU/Linux is a complete software solution using Linux as free operating system (kernel) and free GNU tools on it.

Some definitions

- Free Software
- Free-Software foundation
- Open-Source

Open Source is a newer organization that uses 'Open' instead of 'free' to avoid confusion between free in the expressions
* free beer
* free entrance

Some definitions

- Free Software
- Free-Software foundation
- Open-Source
  - Not only access to the sources

Open source does not only mean access to the sources.

Some definitions

- Free Software
- Free-Software foundation
- Open-Source
  - Not only access to the sources
  - Slight changes compared with Free Software:

It actually gives the same rights as the 'Free software' but with some differences:

Some definitions

- Free Software
- Free-Software foundation
- Open-Source
  - Not only access to the sources
  - Slight changes compared with Free Software:
    - Distribution of modified copies may be restricted

The distribution of modifications can be limited to patches. This does not harm the freedom of changing the source and distributing the modified software but force some respect to the original version of the software, avoiding various slightly different versions to be distributed under the same name and version number.

Some definitions

- Free Software
- Free-Software foundation
- Open-Source
  - Not only access to the sources
  - Slight changes compared with Free Software:
  - GPL is compliant with Open-Source definition

Actually, the most famous license of the 'free software world' (the GPL) is compliant with Open Source principles.

Some definitions

- Free Software
- Free-Software foundation
- Open-Source
- Standards

A standard is not a widely used format or protocol: it is
* Published
* Exist in multiple independent implementations
* Is interoperable between different systems

Ex: IP, HTML, ...
Counter examples: Autocad, Word, Media-Player...

Situation in embedded software world

Situation in embedded software world

- Large amount of processors, micro controllers

There exists a huge amount of different hardware architectures, including various processors or micro-controllers, busses, interfaces, ...

Situation in embedded software world

- Large amount of processors, micro controllers
- Numerous hardware architectures

Embedded systems means usually an architecture that must fits the requirements of the application, no more, no less, and be low cost for large series industrial production. This leads to about as much architectures as embedded applications.

Situation in embedded software world

- Large amount of processors, micro controllers
- Numerous hardware architectures
- Lots of different interfaces

Embedded systems have usually various interfaces (or even about no interface, such as a car alarm). When thinking about a Internet-ready home TV and a router, it is clear that the nature of the interfaces are not exactly the same.

Situation in embedded software world

- Large amount of processors, micro controllers
- Numerous hardware architectures
- Lots of different interfaces
- Real-time, preemptive multi-tasking support required

Most of the time, embedded applications are supposed to react in a given, limited, controlled amount of time (as example an ABS system embedded in a car) and perform multiple tasks at the same time. Real-time services and preemptive multi-tasking is thus required.

Situation in embedded software world

- Large amount of processors, micro controllers
- Numerous hardware architectures
- Lots of different interfaces
- Real-time, preemptive multi-tasking support required
  - → Many proprietary embedded systems

Consequence: the embedded systems world has known a huge amount of different proprietary embedded OS and software suites.

Situation in embedded software world

- Large amount of processors, micro controllers
- Numerous hardware architectures
- Lots of different interfaces
- Real-time, preemptive multi-tasking support required
  - → Many proprietary embedded systems
  - → High cost for a proprietary system maintenance

Because of they were proprietary and non-compatible, the cost of development of drivers for new interfaces, the cost of port to new architectures, ... was not shared among large number of customers, leading to high prices for such dedicated embedded systems.

Situation in embedded software world

- Large amount of processors, micro controllers
- Numerous hardware architectures
- Lots of different interfaces
- Real-time, preemptive multi-tasking support required
  - → Many proprietary embedded systems
  - → High cost for a proprietary system maintenance
  - → Requires efficiency, portability, standardization

So the embedded system we need requires efficiency (low CPU and memory use), portability (sharing the development costs on all the platforms) and standardization (for strategic reasons but also to lower the price)

Situation in embedded software world

- Large amount of processors, micro controllers
- Numerous hardware architectures
- Lots of different interfaces
- Real-time, preemptive multi-tasking support required
  - → Many proprietary embedded systems
  - → High cost for a proprietary system maintenance
  - → Requires efficiency, portability, standardization

→ Embedded Linux

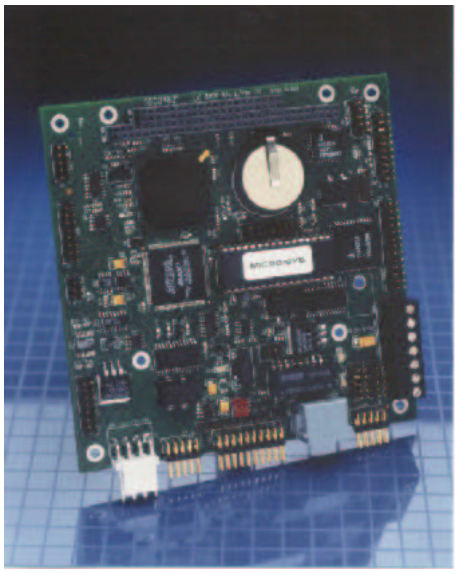
The embedded software suite of choice is Linux, as we will see in the next slides.

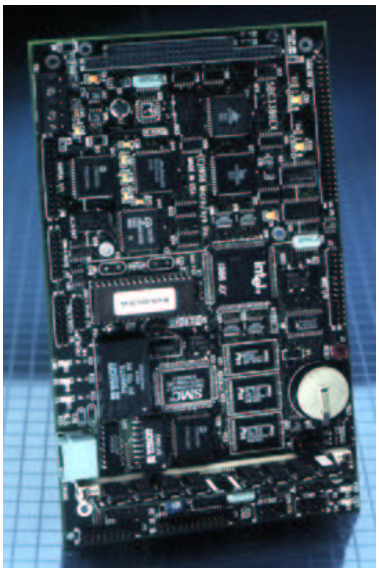
Example of embedded devices: PDAs

Numerous PDA vendors (in Asia) are basing their systems on Linux, with the advantage that huge amount applications (games, web browsers, word processors) are already available for free.

Linux runs on Mips, ARM, ... Cross-compilation is not an issue. Stability of Linux allows the system to go to 'sleep mode' when not used, so the system is actually never rebooting. This is for sure not possible with OS that requires reboots to change settings, install software or suffer from memory leakings...

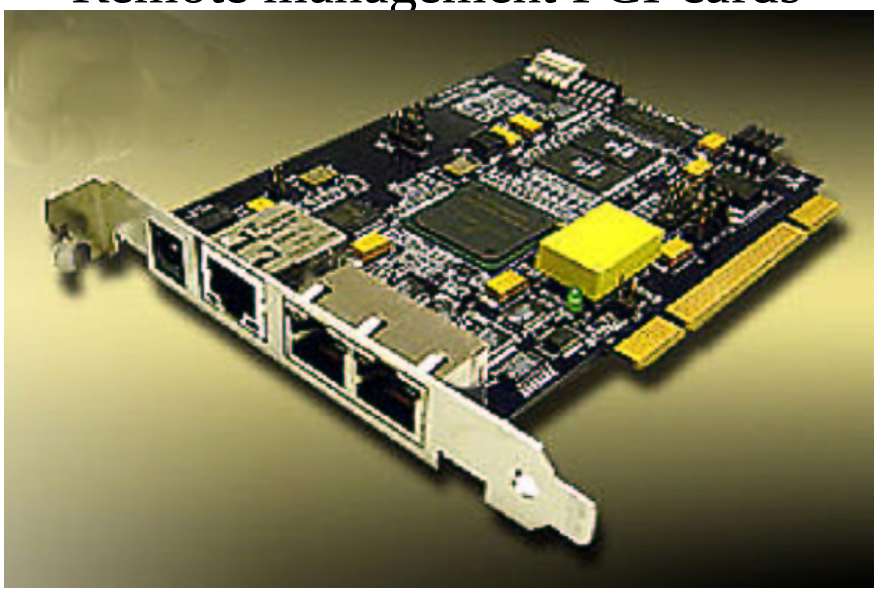
Example of embedded devices: Single board Pcs





Single board PCs are ready for embedded applications. From few EUROs, you can purchase a complete computer, with USD, IDE, Serial, Ethernet interfaces. Linux can be used with only a serial interface as console.

Example of embedded devices:
Remote management PCI-cards



In order to remotely manage hosts running instable unreliable OSes, a remote management card can be used. Often, it just runs a reliable embedded OS: Linux, allowing to remotely reboot the ill computer or access some part of it for remote diagnostic.

Example of embedded devices: Network elements

Terminal servers, routers

IP/DVB Gateways

Network storage devices

Terminal server offers 8 serial console ports plus a modem connection and an Ethernet interface. It offers secured remote network connections (SSH). System fits in 4MB of ROM and runs on 16MB of RAM.

File servers running Linux and booting from a ROM can support huge journalized file systems (300GB) on software raid-5 and recover from a power outage (they don't crash you know) in 27 seconds. This kind of availability makes the user unaware of the server power-cycle: he just notice some network slow-down for some time...

Example of embedded devices: Home devices

Watch

Web screen phones

Digital set-top box

Internet TV

Car MP3 player

VoIP phones

Linux is so CPU-cycles efficient that it fits in a watch. It can work in devices with very different interfaces, such as a phone or an Internet-ready TV.

Why embedded Linux ?

Why embedded Linux ?

- Proved stability, reliability

Uptimes of three years are usual on computers running GNU/Linux or other free OS (FreeBSD). Systems can be updated, softwares can be installed/uninstalled, drivers can be loaded/unloaded without a system reboot.

Groups are even working on ways to upgrade the entire Linux kernel without interrupting the system at all (without reboot) !

The system is not suffering from memory leaking, improper memory protection or uptime limitations.

Why embedded Linux ?

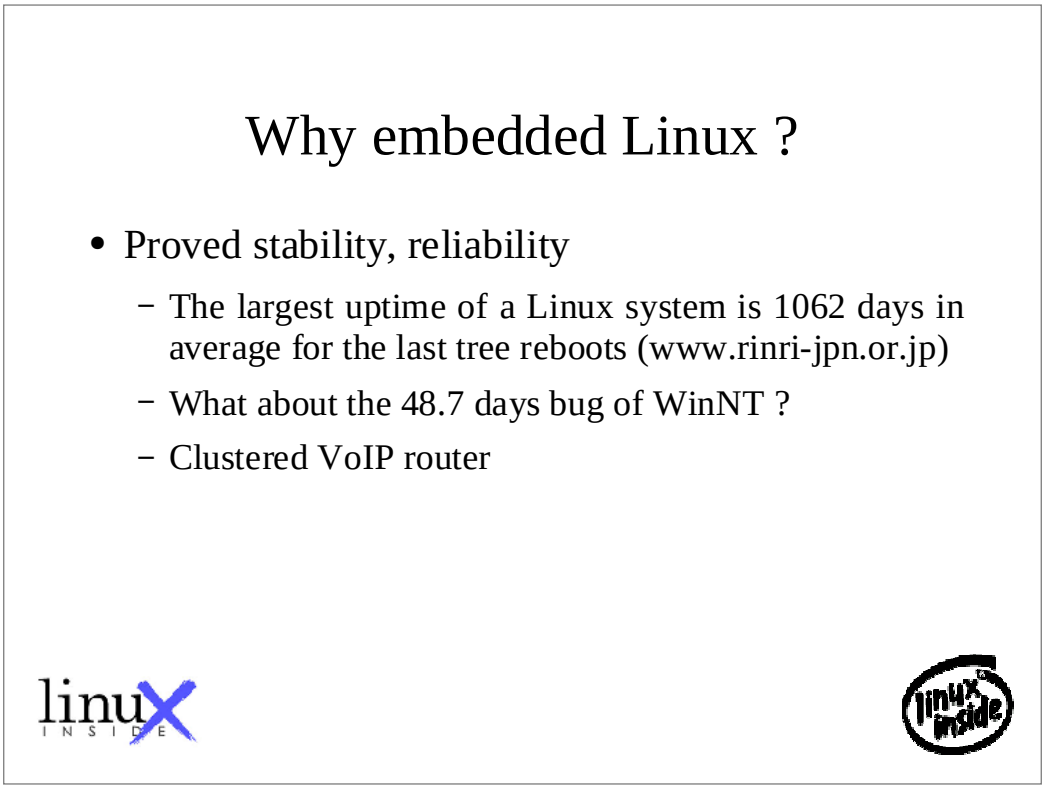
- Proved stability, reliability
  - The largest uptime of a Linux system is 1062 days in average for the last tree reboots (www.rinri-jpn.or.jp)

Largest up-time systems on the Internet (about 4 years of uninterrupted service in average on the last 3 reboots) is achieved by BSD and Linux systems only.

Why embedded Linux ?

- Proved stability, reliability
  - The largest uptime of a Linux system is 1062 days in average for the last tree reboots (www.rinri-jpn.or.jp)
  - What about the 48.7 days bug of WinNT ?

Some years ago, Microsoft recognized an uncorrectable bug (part of the specs actually) in WinNT 4 that prevents such a system to run reliably longer than 48.7 days. Nobody notice, because Microsoft OSes are so unstable and memory-leaking that it is admitted that such a system crashes from time to time. Such a bug would be immediately noticed by Unix-like system administrators, because those systems have typical uptimes of several years.



A Microsoft partner producing VoIP gatekeepers presented in 2000 its product, proud of its stability: it was based on a clustered embedded WinNT system. The justification of the speaker was: 'under this heavy work load, each of the node gets crashed in average once per day. But thanks our cluster architecture, the other node is keeping the services. The crashed node takes less than 3 minutes to reboot, so that the probability of having the two nodes crashed simultaneously is very low, providing a very reliable service.'

Incredible, isn't it ? He was proud of using a buggy, unreliable OS.

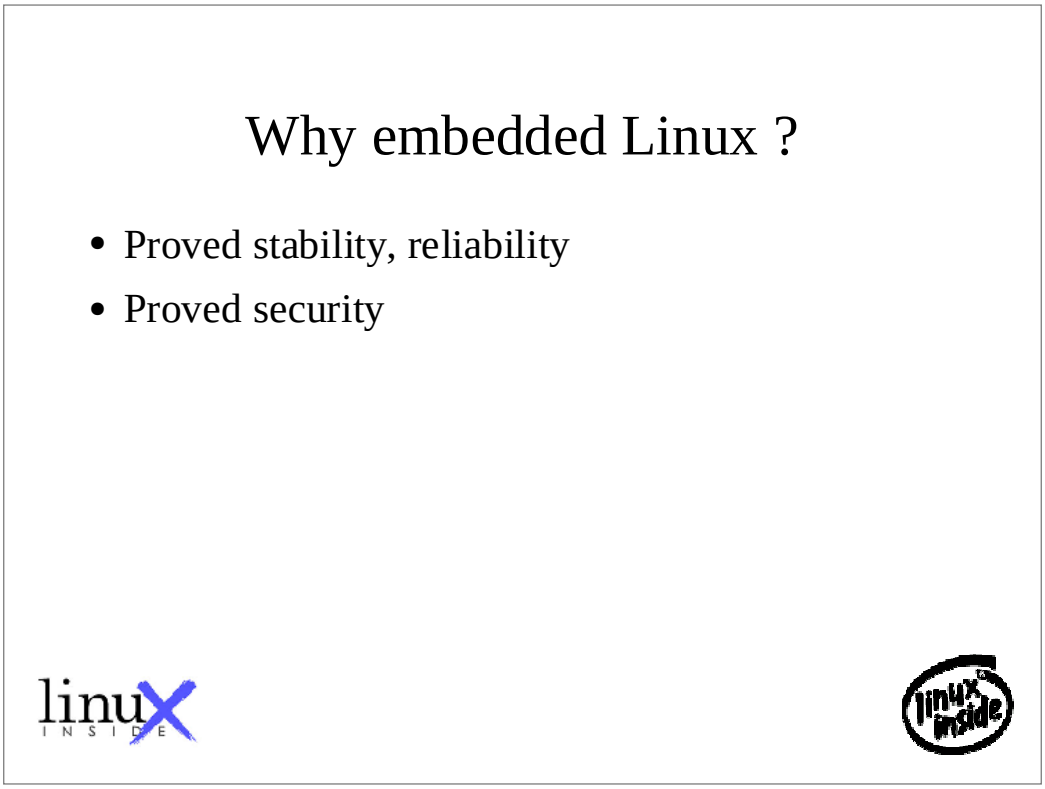
Why embedded Linux ?

- Proved stability, reliability
  - The largest uptime of a Linux system is 1062 days in average for the last tree reboots (www.rinri-jpn.or.jp)
  - What about the 48.7 days bug of WinNT ?
  - Clustered VoIP router
  - Embedded systems are not supposed to be rebooted

Uptimes of three years are usual on computers running GNU/Linux or other free OS (FreeBSD). Systems can be updated, softwares can be installed/uninstalled, drivers can be loaded/unloaded without a system reboot.

Groups are even working on ways to upgrade the entire Linux kernel without interrupting the system at all (without reboot) !

The system is not suffering from memory leaking, improper memory protection or uptime limitations.



Lloyds insurance fees are 30% lower to cover security attack risks on Linux-based systems than on Microsoft ones, as example.

TIS recommends installation of its Gauntlet firewall on BSD or Linux instead of proprietary OS because of the proved and verifiable security of their network stack, among others.

Security fixes are often available within few days on free software, while they are sometimes not made available at all on some proprietary OS.

Why embedded Linux ?

- Proved stability, reliability
- Proved security
- Is ported on about any platform

Linux is ported on ARM, Mips, PowerPC, Sparc, Alpha, Intel 386, 486, Pentium I, Pentium II, Pentium III, Pentium IV, IA 64, lots of micro-controllers, ...

If you have an another platform, port might be already available, or you can port it yourself: you're free !

Why embedded Linux ?

- Proved stability, reliability
- Proved security
- Is ported on about any platform
  - Linux for X-Box (since 3/9/2002)

Examples of port of Linux:

Linux is available since 3/9 on Microsoft's XBox, despite an architecture dedicated to prevent execution of softwares not approved by Microsoft.

Why embedded Linux ?

- Proved stability, reliability
- Proved security
- Is ported on about any platform
  - Linux for X-Box
  - Linux for Cisco routers

Belgacom has ported Linux on Cisco's proprietary routers architecture to enjoy the functionalities and reliability of Linux.

Why embedded Linux ?

- Proved stability, reliability
- Proved security
- Is ported on about any platform
  - Linux for X-Box
  - Linux for Cisco routers
  - Linux for Palm, Pocket PCs, ...

Linux has been ported on hardware that have not even the required functionalities to support Linux: Linux is a demand-paging, virtual memory OS that uses memory protection and dynamic address translation provided by the underlaying hardware (state-of-the art in operating systems). Palm PDAs hardware does not provide those services, but Linux is however available on it.

Linux runs also on Pocket PCs such as IPaq.

Why embedded Linux ?

- Proved stability, reliability
- Proved security
- Is ported on about any platform
  - Linux for X-Box
  - Linux for Cisco routers
  - Linux for Palm, Pocket PCs, ...
  - Linux as first OS ready for IA64

Intel gave to the Linux kernel development community the specifications and a simulator of the future IA64, the 64 bits Intel processor.

On the first day this processor was available, Linux was ready for it and ran. It was the only OS available on IA64 at that time.

Why embedded Linux ?

- Proved stability, reliability
- Proved security
- Is ported on about any platform
- Exists a real-time kernel patch

The Linux kernel is not especially designed for real-time services. However, a patch exist in order to replace the kernel scheduler by a real-time one.

Why embedded Linux ?

- Proved stability, reliability
- Proved security
- Is ported on about any platform
- Exists a real-time kernel patch
  - Non RT kernel gives real-time services

But the unpatched kernel offers already pseudo-real-time system calls, such as timers and watch-dog support.

Why embedded Linux ?

- Proved stability, reliability
- Proved security
- Is ported on about any platform
- Exists a real-time kernel patch
  - Non RT kernel gives real-time services
  - Non RT kernel has 100 μs accuracy

Our experience using Linux in real-time applications is that non patched kernel can respect timings so tiny as 0.1 ms, which is already very good result. In comparison, Windows systems have an accuracy of 10ms.

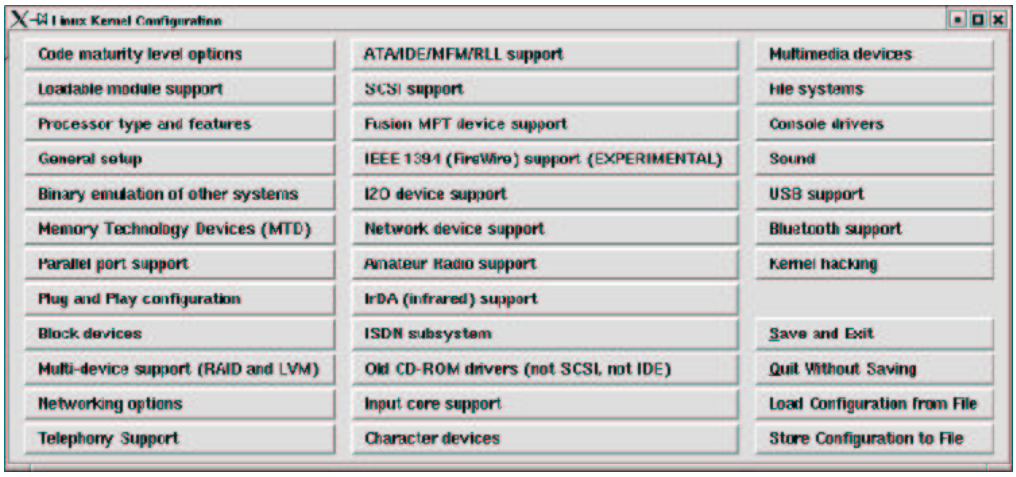
A car at 120 Km/h rides 30cm in 10ms while it rides only 3mm in 0.1 ms. The difference in the case of embedded security systems is obvious.

Why embedded Linux ?

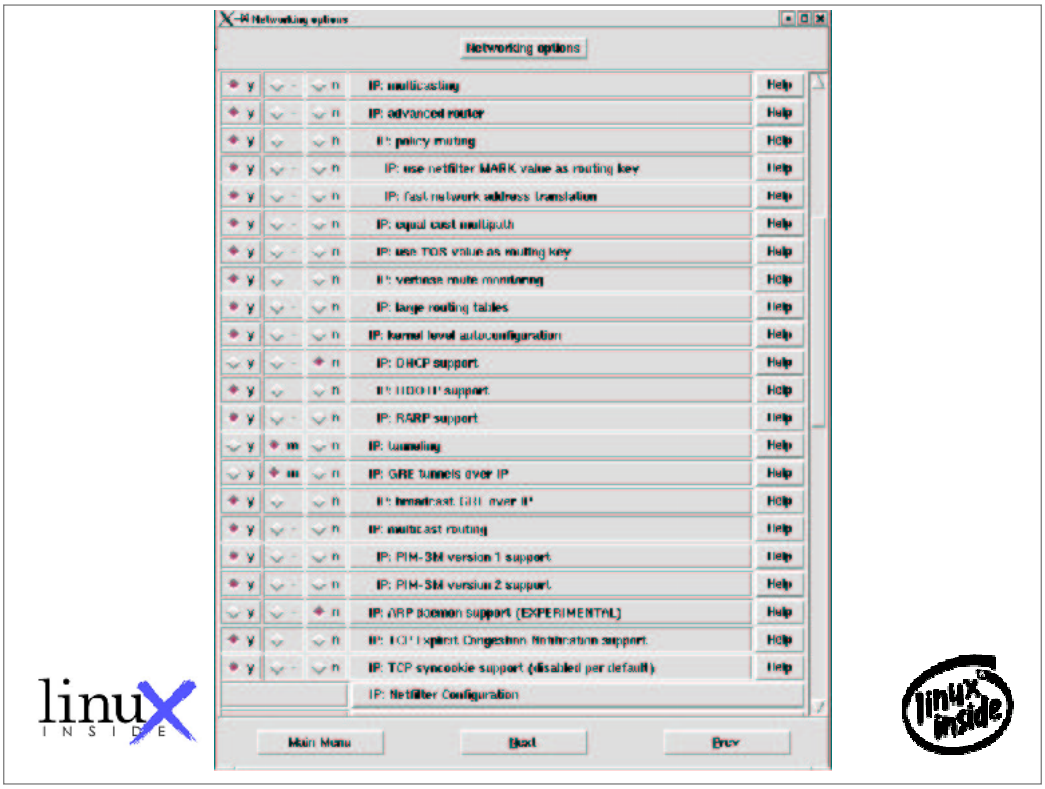
- Proved stability, reliability
- Proved security
- Is ported on about any platform
- Exists a real-time kernel patch
- Can be tailored to match specific embedded needs

Production of a dedicated kernel supporting exactly the services required by an embedded application is provided by Linux from day one.

Why embedded Linux ?



Even a graphical interface is provided to choose what should be part or not of a given kernel. Production of a dedicated kernel is easy to do.



Options are numerous: under the IP stack option, lots of elements are available for choices. Please note that this snapshot is partial. Other options are available above and below this partial list.

Why embedded Linux ?

- Proved stability, reliability
- Proved security
- Is ported on about any platform
- Exists a real-time kernel patch
- Can be tailored to match specific embedded needs

An OS supporting Ethernet, IPv4, IPv6, Ethernet, PPP, Serial, PCI, Bootp, DHCP, compressed ROMFS, USB, JFS, IDE, ... fits in 360 KB.

A shareware using '.NET' requires 300MB of libs...

I build a Linux kernel image for an embedded project choosing exactly what I needed. The image of the OS is 360KB. It perfectly fits in an embedded system ROM.

Compared with this, it shows the importance of the choice of an OS:

> Hello. Je vous trouve très gentil avec MS. Moi, je n'y arrive plus. En fait, je suis programmeur VB depuis QB 2.0. Je suis passé par QB 2.0, QB 4.5, MSPDS 7.0, VB DOS 1.0, VB WIN 2.0, VB WIN 4.0, VB WIN 5.0, VB WIN 6.0.... A l'époque, j'avais fais un peu de C, de C++ et de l'assembleur (j'oublais : un peu de GWBASIC aussi!) sans pouvoir décrocher de la famille du basic à cause de son enorme facilité de dev rapide et ce malgré ses problèmes de mémoire et de vitesse.
>
> J'ai attendu longtemps d'avoir un VB7 et aujourd'hui, je peux vous dire que je suis très décu. Au point que je viens de changer de langage en choisissant borland c++ 6.0 Builder. J'aime pas delphi, c'est pas forcement logique mais surtout viscéral.
>
> Pourquoi suis-je décu ? J'ai bien sur reçu les bétas de .NET, j'ai fais un peu de dev dessus puis j'ai arrêté, non sans prendre un abonnement MSDN pour être sur d'avoir la version finale des qu'elle sort.
>
> Et des qu'elle est sortie, je me suis mis sur un projet perso plutôt technique et complexe - vu la force de .net, je me suis dis que je pouvais mettre le paquet : un gros logiciel d'analyse graphique pour la bourse
>
> Très bien. Je commence et à part quelques problèmes de clarté dans le langage à propos des déclarations valeur/référence, tout va bien.... Et j'apprécie énormément le comilateur intégré qui permet de recompiler du langage VB.NET depuis un exécutable .NET - Ca permet à mes utilisateurs de programmer eux même des courbes graphiques et en plus ça compile.
>
> Seulement, quand je commence à afficher une dizaine de courbes graphiques calculées en temps réel, je me rend compte que le truc va franchement pas assez vite et qu'il sera rapidement dépassé, je cherche des solutions, je pleure sur les NG de microsoft, j'obtient quelques réponses que j'applique à mon programme, ça s'améliore un peu mais finalement ça me convient toujours pas. Pour palier définitivement à ce problème, j'utilise la bibliothèque de compatibilité .NET / VB6 et je dessine mes graphiques en utilisant DirectX8. La plus de problèmes, ca booste. La lenteur provient donc bien de GDI+. Bon, je me sens pas très bien quand même parce que ça m'oblige à distribuer le framework : 20 méga, DirectX 8 : environ 10 mega, mon programme : 300ko.
>
> Par acquis de conscience, je fais un test d'installation de tout ça sur un windows 98 formaté. Je copie le necessaire : .NET framework redist, DirectX 8.0 et mon programme.
>
> Je double clique sur .net framework installation et vlan premier message : il faut IE6 pour installer le truc. Resultat: Je télécharge IE6 : 80 mega
> Je l'installe et j'installe le framework: premier reboot. J'installe DirectX :
> deuxieme reboot. Je clique sur mon programme pour le lancer :
> il manque MDAC (pour les bases de données). je télécharge MDAC 2.6 et j'installe : troisième reboot. Je clique a nouveau sur mon programme, il manque MS-JET 4.0 (Gestion des bases de données Access) : Je télécharge : 5 MEGA. Je reclique sur mon programme : il manque les bibliothèques de compat VB6 (pour directx) fournies avec VS.NET mais pas avec le framework. J'installe les bibliothèques et enfin mon programme marche. Mais sacrée galère quand même.
>
> Pour faire tourner un programme de 300 ko sur un poste w98 vierge, j'ai du installer environ 120mega de programmes compressés. C'est pas mal pour un programme que je veux distribuer en shareware !!! Absurde !!!
>
> Si le framework est bien en substance, il se traine des tares monstres. C'est à dire qu'il faut quasiment distribuer une version .net de windows pour un programme de 300ko. De plus, il est franchement trop lent. Si sur de nombreux points, il est plus rapide que VB6, je suis persuadé que sur d'autre il l'est moins. Tout ceci, m'a dégouté et me fait changer de langage, avec regrets mais sans choix.
>
> J'ai aussi testé ASP.NET et je peux vous dire que mes programmes fait en ASP 2.0 était compilé et tournaient très vite car j'utilisais ASP uniquement comme passerelle de quelques lignes pour lancer des DLL ActiveX. De ce point de vue, ASP.NET ne révolutionne rien pour moi. La compilation existait déjà bien avant. Concernant les WEB Controles, ils n'ont aucun interet, si ce n'est de faire ramer l'application web. En effet, ces contrôles se comportent comme les contrôles d'une applications windows classique : ils déclenchent des événements qui sont immédiatement renvoyés vers le serveur mais avec l'énorme problème que cela passe par internet et que ca ralentit grave. En intranet, ça peut être interessant, mais sur internet : zéro. On en revient à la programmation classique de site web, mais avec l'avantage évident de pouvoir programmer avec de vrais langages : VB.NET, C# et bientôt DELPHI.NET, c'est quand même bien pour ça, ces langages sont nettement plus évolués que les anciennes versions d'ASP. Par contre, programmer un site web avec .NET, c'est pas une sinécure car 1) VS.NET rame lourdement et l'alternative WEBMATRIX n'en est pas une car elle est trop limité : pas d'intellisense, pas de mise en forme de la syntaxe, etc. Ensuite, il faut au minimum windows xp pro ou windows 2000 et bien sur IIS version 5 minimum. Auparavant, on pouvait créer son site web avec PWS (personal web server) gratuitement et sous windows 98. Donc financièrement, c'est pas la même histoire non plus. En revanche, le concept des web services semble très intéressant pour les applications distribuées.
>
> En conclusion, à l'heure actuelle il apparait évident que .NET n'est pas adapté au developpement d'applications windows critiques, rapides, portables entre différentes versions de windows, à des distributions grand public par internet sous forme de sharewares ou freeware, etc
>
> Il apparait, par contre, etre une évolution sérieuse du langage ASP en permettant la programmation sous différents langages de sites web et integre les controles web pour des applications intranet. Ou encore des applications distribués via les services web.
>
> C'est donc plutôt un produit qui vise plutôt les entreprises. L'installation du framework sur tous les postes serait un véritable plus dans de nombreux cas malgré l'importante mise à jour necessaire. Si l'on devait faire une comparaison avec Java, je dirais que si Java n'avait pas la portabilité qu'il a, il serait écrasé par par le trio Visual Studio.NET / .NET Framework / C# mais comme ce n'est pas le cas, les choses sont différentes. Dans le cas ou l'on le poid du déploiement des outils ne pose pas de problème et que la plate forme cible est windows/internet, .net reste l'outil de premier choix. Dans le cas contraire, il faut utiliser Java sans discuter.
>
> Quand à moi, je respecte cette façon de voir, et j'ai choisi borland Builder c++ car il dispose d'outils rad inexistants ailleurs que chez microsoft. En effet, BC++ Builder permet de construire une appli de la même façon qu'avec vb.net. La bibliothèque fournie avec Borland C++Builder version entreprise est plus riche que celle de .net. par ailleurs, les applications produites avec Borland C++Builder sont rapides et portables (linux,windows95,98,etc). Il est d'ailleurs fort probable que je change de plate forme pour linux pour des raisons économiques: la dependance microsoft revient chere (notament à cause des licences)
>
> Même si j'ai évité les aspects purement techniques, j'espère avoir apporté un peu de lumière sur .net.
>
> A+

Why embedded Linux ?

- Proved stability, reliability
- Proved security
- Is ported on about any platform
- Exists a real-time kernel patch
- Can be tailored to match specific embedded needs

The YOPY kernel+base applications fits in 24 MB

(OS, X-Window, base applications...)

As example, the commercial PDA called Yopy, has a total memory of 64MB. The OS, base applications such as graphical interface, multimedia player, web browser, e-mail client, games, shell, ... fits only in 24 MB of RAM.

Why embedded Linux ?

- Proved stability, reliability
- Proved security
- Is ported on about any platform
- Exists a real-time kernel patch
- Can be tailored to match specific embedded needs
- Is maintained, ported by the community

Linux was created in 1991. The evolution of this OS was so fast, due to the Internet community support, that it supplanted Hurd (the free OS developed by the FSF) and it evolved to the best OS available, forcing IBM to port Linux to all its platforms from the desktop to the mainframe.

(Some NASDAQ IBM mainframes are running Linux)

Why embedded Linux ?

- Proved stability, reliability
- Proved security
- Is ported on about any platform
- Exists a real-time kernel patch
- Can be tailored to match specific embedded needs
- Is maintained, ported by the community
- Is free

Meaning that for companies that sells hardware (not software), they have the total freedom to access the embedded OS source code, to correct it, adapt it, tune it and use it on as many systems as they want for free, receive for free updates, bug fixes, security fixes, ...

Why embedded Linux ?

- Proved stability, reliability
- Proved security
- Is ported on about any platform
- Exists a real-time kernel patch
- Can be tailored to match specific embedded needs
- Is maintained, ported by the community
- Is free
- Is very efficient in memory and CPU usage

Tests made by Oracle and newspapers shows that on the same hardware, Linux is in average about 30% more efficient than Microsoft OS.

A Linux-based router running G703 2Mb/s line, plus IEEE 802.1q vlan tagging on a fast-Ethernet line, plus SDLS on PPPoE, plus firewalling, routing, Network Address Translation, web proxy service, file and graphical desktop service plus other services (SNMP, NTP, FTP, NFS, ...) fits in 128MB of ram. No swap is required.

Why embedded Linux ?

- Proved stability, reliability
- Proved security
- Is ported on about any platform
- Exists a real-time kernel patch
- Can be tailored to match specific embedded needs
- Is maintained, ported by the community
- Is free
- Is very efficient in memory and CPU usage
- Is designed to work without graphical interface

Linux can be configured to offer all services (including remote GUI) on system having no graphical interface ! Embedded devices, such as routers, MP3 players, set-top boxes, car radios... can be fully accessed, maintained, debugged through a simple Ethernet or serial interfaces.

Strategic considerations

- Anti-competitive proprietary OSes

Proprietary OSes can have anti-competitive behaviors such as hiding APIs, changing implementations from update to update to prevent competitors software to be supported, hide some hardware specifications to prevent competitor's product to use it, not support some interface that did not pay an entrance fee, integrate undocumented services to their 'OS' to give advantage to their own software against concurrents.

Strategic considerations

- Anti-competitive proprietary OSes
  - The Office Suite example

Microsoft finally admitted that hidden APIs were part of Windows OS suite to give advantage to MS-Office. Too late: there isn't a commercial alternative to MS-Office on Windows any more.

Strategic considerations

- Anti-competitive proprietary OSes
  - The Office Suite example
  - The web browser example

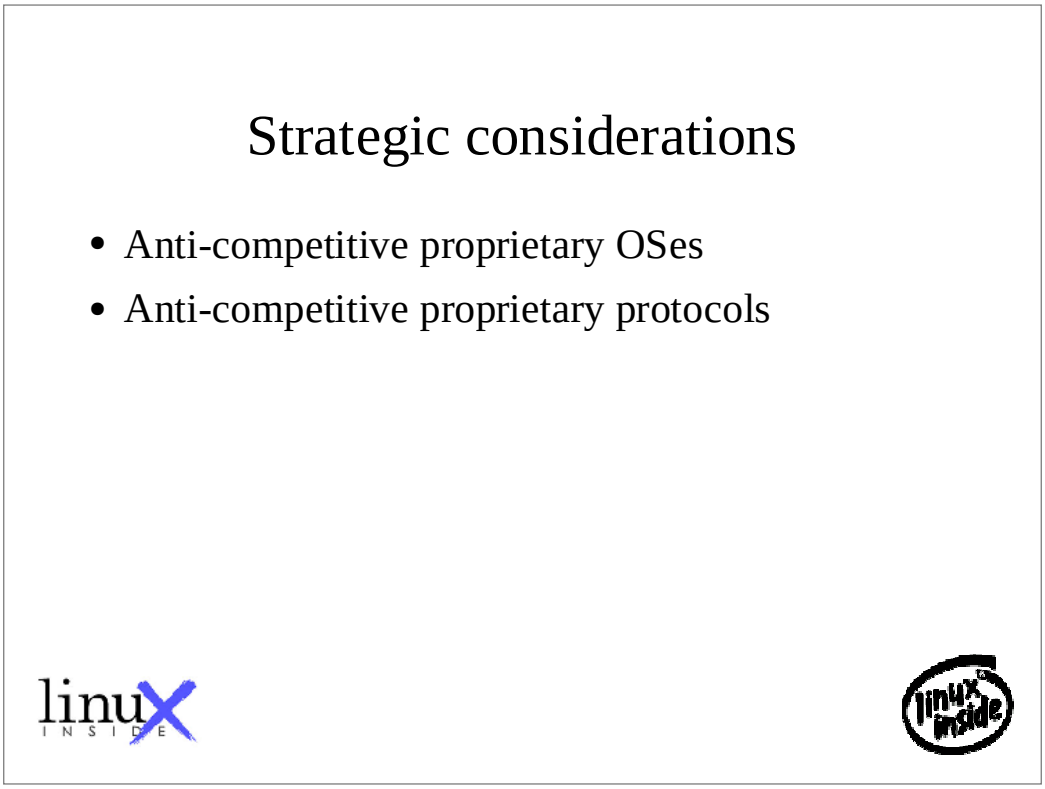
The market of commercial web browsers is out. Microsoft integrated this functionality in its OSes against legal decisions (Microsoft was already condemned before launch of Windows-95 for this). Developing a commercial product based on Windows means that you play a game against an opponent that rules the game, changes the rules during the game & does not respect them.

Strategic considerations

- Anti-competitive proprietary OSes
  - The Office Suite example
  - The web browser example
  - The Palm example

Palm since 2000 does not support anymore synchronization between its PDAs and laptops by IR interfaces because Microsoft changed the way this device was handled by Windows since Win2000 and refuse to disclose the way to use it.

Knowing that Microsoft decided to enter the PDA market, it's not strange to see them refusing such an information to their main concurrent on that market. Such a behavior has a name: it abuse of dominant position to extend monopoly to other markets. It is strictly illegal but when it will be ruled, Palm will be out of the market since years. Take this as example and avoid any link with proprietary OSes for your embedded business, if you don't want to be out in short term.



Same applies to protocols. As soon as a proprietary protocol is used, the dependency to the legal owner of the protocol is total. That's why telecommunication standards have always been published and interoperable between various manufacturers: GSM are using a standard to communicate, TVs are all using same published process to reproduce pictures, phones of any trademark are usable on public phone network, success of the Internet is based on published IP, TCP, UDP, HTML standards, success of CDs is due to the publication of the standard by its creator.

Strategic considerations

- Anti-competitive proprietary OSes
- Anti-competitive proprietary protocols
  - Media-player

Media-player is a video signal streaming format that is NOT published, not interoperable, forces the customer to purchase proprietary OS, forces the service provider to purchase proprietary streamer... It's exactly the opposite of a standard and is dangerous because it concentrate control on multi-media services in the hands of a single monopolist. Could we accept a single company to control totally the telecommunication market ?

Strategic considerations

- Anti-competitive proprietary OSes
- Anti-competitive proprietary protocols
  - Media-player
  - CIFS

Instead of supporting standardized file transfer protocols such as FTP, NFS, HTTP file transfer protocol, ... Microsoft preferred to use the CIFS published standard. Because this allowed interconnection between Windows-based system with other systems running CIFS services, Microsoft changed progressively the CIFS implementation so that it is no more a standard. Even the famous open-source CIFS implementation that follows those changes is forbidden to use by Windows licenses. Microsoft hates interoperability, prevents it by changes in its proprietary protocols and if it's not enough, prohibit interoperability by license terms.

Strategic considerations

- Anti-competitive proprietary OSes
- Anti-competitive proprietary protocols
  - Media-player
  - CIFS
  - ...

Strategic considerations

- Anti-competitive proprietary OSes
- Anti-competitive proprietary protocols

=> Avoid using proprietary elements under control of (potential, future, actual) concurrents !

Integration of proprietary elements in an embedded system puts lethal dependency on potential or actual concurrents. Examples are numerous: in mid or long-terms, this will put you out of business because of illegal anti-competitive behaviors. More than that, EC directive demand publication of telecommunication standard when used on public network and targeting public audience. Use of proprietary protocols or systems supporting only proprietary protocols is to be avoided by principle and by law.

Strategic considerations

- Anti-competitive proprietary OSes
- Anti-competitive proprietary protocols

=> Avoid using proprietary elements under control of (potential, future, actual) concurrents !

Counter examples:

Strategic considerations

- Anti-competitive proprietary OSes
- Anti-competitive proprietary protocols

=> Avoid using proprietary elements under control of (potential, future, actual) concurrents !

Counter examples:

- Sony's success with Sony PlayStation II

Sony is not out of the game console business because it has chosen to build it's embedded system on non proprietary system and they develop their game on Linux. They have the control of their hardware and software platform. They don't depend on concurrent's proprietary parts.

Strategic considerations

- Anti-competitive proprietary OSes
- Anti-competitive proprietary protocols

=> Avoid using proprietary elements under control of (potential, future, actual) concurrents !

Counter examples:

- Sony's success with Sony PlayStation II
- Ericsson & Nokia using Java technology on GSM

Ericsson and Nokia understood that the proposed OS for GSMs by Microsoft cannot be used by them. If they take that alternative, Microsoft will sell all GSMs within next few years. They have no choice but using standard systems and protocols not under control of a proprietary software vendor: they have chosen Java as software platform for their next generation GSMs.

Strategic considerations

- Anti-competitive proprietary OSes
- Anti-competitive proprietary protocols

=> Avoid using proprietary elements under control of (potential, future, actual) concurrents !

Counter examples:

- Sony's success with Sony PlayStation II
- Ericsson & Nokia using Java technology on GSM
- ...

Strategic considerations

Strategic considerations

- Stay hardware independent: use a multi-platform OS

In order to be ready for new emerging hardware and keep the possibility to change of hardware if useful, in order to be in good position for the negotiation with hardware manufacturer, the choice of an OS and software suite that is portable must be done.

Strategic considerations

- Stay hardware independent: use a multi-platform OS
- Stay vendor independent: use multi-vendor OS

In order to be vendor-independent, choose an OS that is sold by multiple distributors. The same product (Linux) is proposed in more than 200 different distributions. Prices are variable:

* RedHat Linux professional is around 275 USD.
* Suse Linux professional is around 40 EUR.
* Some distributions are available gratis.

Strategic considerations

- Stay hardware independent: use a multi-platform OS
- Stay vendor independent: use multi-vendor OS
- Stay OS independent: use standard API, libs, tools, ...

It is possible with a bit of attention to develop software that does not contain Linux-specific code. Ex: Ansi-C, C++ with TCL/Tk graphical interface based on GLIB, or Perl, Java, ... applications can run on as many platforms as Linux, FreeBSD, NetBSD, OpenBSD, Solaris, MaxOS X, Win95, Win98, Win2000, WinNT3.51, WinNT4, WinXP, HP-UX, IRIX, AIX, True64, Hurd, ...

This way, the project remains highly independent of a specific OS. If some Linux-specific code is required, locate it in a documented specific part of the code (OS adaptation layer module) so that port will be easier.

Strategic considerations

- Stay hardware independent: use a multi-platform OS
- Stay vendor independent: use multi-vendor OS
- Stay OS independent: use standard API, libs, tools, ...
- Keep business freedom: no export regulations in licenses

Keep your business market as open as the world is. In some proprietary OS licenses, there are exportation restrictions. There is no guarantee the list of banned countries will not change, so using those softwares in an embedded application gives the control of a foreign country on the market you can work on.

Some US exportation regulations also limit the functionalities of software, even for European Union: security is treated differently for exported software than for local ones. Do we have to accept that products we sell must be less secure than concurrent ones ? Is this difference limited to security ? Will it remain limited to security issues ?

Strategic considerations

- Stay hardware independent: use a multi-platform OS
- Stay vendor independent: use multi-vendor OS
- Stay OS independent: use standard API, libs, tools, ...
- Keep business freedom: no export regulations in licenses
- Keep your customers base secret: no mandatory OS registrations

In order to protect your business against unfair competitors, it is important to keep secrecy about your customers. OSes or other proprietary software that requires on-line registration shows to (potential) competitors the list of your embedded application customers, gives as much information as the proprietary OS vendor wants to know.

Strategic considerations

- Stay hardware independent: use a multi-platform OS
- Stay vendor independent: use multi-vendor OS
- Stay OS independent: use standard API, libs, tools, ...
- Keep business freedom: no export regulations in licenses
- Keep your customers base secret: no mandatory OS registrations
- Be ready for new emerging platforms

Evolution to new emerging platforms, such as Hurd or L4 is not a problem when using standard APIs and programming languages.

New OSes such as Hurd or L4 are using the same standards: Posix processes and inter-process communication mechanisms, support of the GLIB library and Ansi-C programming language. All other tools, such as other programming language compilers or interpreters are then portable on those new OSes, as well as graphical interfaces and other applications. In other words, if you develop on Linux, you're ready for OSes that are not yet available.

Strategic considerations

- Stay hardware independent: use a multi-platform OS
- Stay vendor independent: use multi-vendor OS
- Stay OS independent: use standard API, libs, tools, ...
- Keep business freedom: no export regulations in licenses
- Keep your customers base secret: no mandatory OS registrations
- Be ready for new emerging platforms
- Avoid uncontrolled licenses prices caused by monopolies

By choosing proprietary tools (OS, programming language, libraries, protocols, formats, ...) provided by a single vendor, even if the first license price is low, your business becomes dependent of a single provider, meaning that the future prices for licenses or services can raise as high as the monopolist decides. You have no way to change of provider (it's unique) but to port your application to an another proprietary system, or choose an open platform, interoperable with other systems, keeping you in good situation for the negotiation of software and services prices.

Linux is currently the most mature platform giving this kind of guarantee: it's free and multi-vendor.

Strategic considerations

- Stay hardware independent: use a multi-platform OS
- Stay vendor independent: use multi-vendor OS
- Stay OS independent: use standard API, libs, tools, ...
- Keep business freedom: no export regulations in licenses
- Keep your customers base secret: no mandatory OS registrations
- Be ready for new emerging platforms
- Avoid uncontrolled licenses prices caused by monopolies
- Keep technology control: Access to all APIs, adapt the kernel...

In order to be able to develop the best embedded applications, access to all the underlaying sources is a must: depending on the kind of application, slight changes to the kernel or some part of it (IP stack, real-time scheduler, ...) will make the difference between your product and concurrent's ones. If the concurrent is using proprietary closed tools, you have key advantage on it. If the concurrent is the vendor of a closed proprietary software suite that you have chosen, you are the looser.

Strategic considerations

- Stay hardware independent: use a multi-platform OS
- Stay vendor independent: use multi-vendor OS
- Stay OS independent: use standard API, libs, tools, ...
- Keep business freedom: no export regulations in licenses
- Keep your customers base secret: no mandatory OS registrations
- Be ready for new emerging platforms
- Avoid uncontrolled licenses prices caused by monopolies
- Keep technology control: Access to all APIs, adapt the kernel...
- Keep up-to-date: Research, high education, new protocols...

New standards (such as Internet protocols, new data formats, ...) are developed by consortiums or work groups such as the IETF. To develop new prototypes of protocols implementations, access to the entire kernel source is required. That's why new protocols, new data formats are developed as example by IETF members on Solaris, BSD and Linux. That's why Linux implemented IPv6 three years in advance on other proprietary systems. Linux also implements already IGMPv3, lots of encryption mechanisms and security features unknown to other proprietary systems. Linux is used in universities, research centers... So it is up-to-date, state of the art OS.

Strategic considerations

- Stay hardware independent: use a multi-platform OS
- Stay vendor independent: use multi-vendor OS
- Stay OS independent: use standard API, libs, tools, ...
- Keep business freedom: no export regulations in licenses
- Keep your customers base secret: no mandatory OS registrations
- Be ready for new emerging platforms
- Avoid uncontrolled licenses prices caused by monopolies
- Keep technology control: Access to all APIs, adapt the kernel...
- Keep up-to-date: Research, high education, new protocols...
- Avoid OS that has a foreign intelligence agency back-door in it

It is well known and has been admitted that Windows systems contains a back-door allowing US intelligence services to enter any computer running this kind of software and connected to a network. Development of embedded applications on such a system is dangerous for strategic reasons: let's imagine all TV sets, all phones, all cars in Europe running this kind of software. What about our independence, our freedom of choice, especially if they diverge from those of US ?

More than that, those kind of OS might be in short or mid-term forbidden by EU authorities for obvious security reasons.

Strategic considerations

- Stay hardware independent: use a multi-platform OS
- Stay vendor independent: use multi-vendor OS
- Stay OS independent: use standard API, libs, tools, ...
- Keep business freedom: no export regulations in licenses
- Keep your customers base secret: no mandatory OS registrations
- Be ready for new emerging platforms
- Avoid uncontrolled licenses prices caused by monopolies
- Keep technology control: Access to all APIs, adapt the kernel...
- Keep up-to-date: Research, high education, new protocols...
- Avoid OS that has a foreign intelligence agency back-door in it
- Avoid OS that are forbidden by states or administrations

So when choosing proprietary OS or software tools for an embedded application, it must be known that some states or organizations are already banning some systems. As example, German army bans use of Microsoft systems in military applications. Other countries, such as China, Peru, UK are following the same direction.

Strategic considerations

- Stay hardware independent: use a multi-platform OS
- Stay vendor independent: use multi-vendor OS
- Stay OS independent: use standard API, libs, tools, ...
- Keep business freedom: no export regulations in licenses
- Keep your customers base secret: no mandatory OS registrations
- Be ready for new emerging platforms
- Avoid uncontrolled licenses prices caused by monopolies
- Keep technology control: Access to all APIs, adapt the kernel...
- Keep up-to-date: Research, high education, new protocols...
- Avoid OS that has a foreign intelligence agency back-door in it
- Avoid OS that are forbidden by states or administrations
- ...

Conclusion

Conclusion

- Linux gives you all strategic guarantees

Conclusion

- Linux gives you all strategic guarantees
- Linux is efficient, reliable, tailorable, portable, up-to-date, secure, fast evolving, widely supported, real-time ready, embeddable, low-cost, implement standard protocols, supports portable APIs, use few memory, does not require GUI...

Conclusion

- Linux gives you all strategic guarantees
- Linux is efficient, reliable, tailorable, portable, up-to-date, secure, fast evolving, widely supported, real-time ready, embeddable, low-cost, implement standard protocols, supports portable APIs, use few memory, does not require GUI...

Linux is the OS of choice

for embedded applications

Links relative to embedded Linux

- http://embedded.linuxjournal.com
- http://www.embedded-linux.org
- http://www.embeddedlinux.com
- http://www.embeddedsys.com
- http://www.realtimelinux.com

Links to embedded Linux products

- http://www.lineo.com
- http://www.yopy.com
- http://www.cyclades.com
- http://www.b2c2.com/products/homestreamer.html
- http://www.penguincomputing.com
- http://hippoinc.com
- http://www.ch1.com
- http://www.opentv.com

Embedded systems: strategic issues

http://BFrere.net/linuxdays2002

Questions ?

Embedded software: strategic issues