新情報家電の創造
実システム創造型プロジェクト

宗像尚郎

株式会社　ルネサスソリューションズ

電気通信大学　2012-10-2

自己紹介

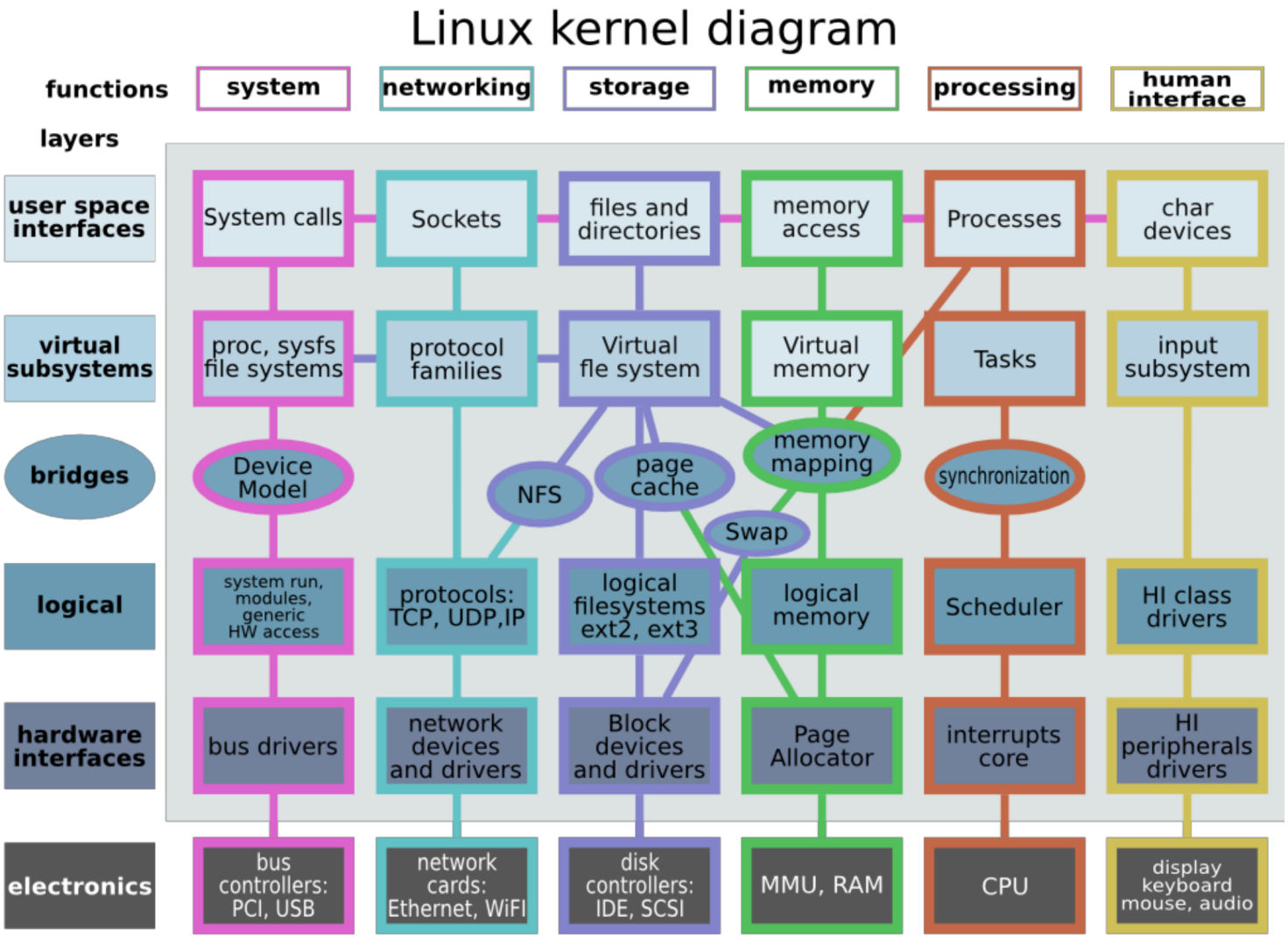
- ルネサスという日本の半導体の会社で働いています。
- Linux Foundation CE[1] ワークグループの理事メンバーで, アーキテクチャ分科会の副議長を務めています。
- LF/CEWG の LTSI[2] プロジェクトの創始メンバーです。
- LF の色々な技術会議の企画委員を務めています。
- 会社では社内のリナックス開発者がオープンソースの開発コミュニティへパッチを投稿するのを支援しています。
- デジタルテレビ、スマートフォン、カーナビ、ブルーレイプレイヤーなどを開発する世界のメーカーの開発支援も行っています。

[1] CE = consumer electronics
[2] LTSI = Linux Foundation の産業界向け長期安定カーネル開発プロジェクト

本日の話題

1. コミュニティによるオープンソース開発の実態
   - リナックスカーネルとはどんなものか
   - リナックスカーネルはどのように開発されているのか
2. 家電業界のオープンソースへの取り組み
   - 賢者は歴史に学ぶが、愚者は自ら失敗するまで学ばない
   - RTOS と Linux のソフトウエアの制御思想の決定的な違い
   - 組み込み機器へのリナックス活用、9つのヒント
3. 持続的イノベーション、破壊的イノベーション
   - トップランナーの リスクを認識する
   - 日本産業界が直面する課題と破壊的リスク要因

リナックスカーネルのハイアラーキー構造



カーネル統計情報 (= 世界最大のオープンソース)

Table : Statistics of Linux kernel 3.5.3 (released 2012.8.26)

number of code lines	15,598,058
number of files	39,097
number of registered maintainer	1,255
number of newly added patches	9,534
update cycle	every 75 - 80 days
configuration items	5,386
history of development	over 20 years

リナックスカーネルの開発に参加してみませんか ?

プログラミングの世界で世界の頂点を目指したいと考えるのなら、オープンソースプログラムを 外して考えることはできないでしょう。中でもリナックスカーネルはコードサイズ、開発者数、リリース間隔、新機能への対応などあらゆる面で圧倒的な世界一の規模となっています。リナックスカーネルの開発に参加 することは、世界のトッププログラマーが競うオリンピックに参加するようなエキサイティング な体験であることは間違いありません。

カーネルソースコード (3.6-rc7) 概況

Directory	Sub-dir	file[3]	line[4]	proportion
Documentation		2,727	7,134	0.05%
arch		17,945	2,422,641	16.75%
	x86	4,582	645,659	4.46%
	ARM	1,229	242,850	1.68%
block		68	27,711	0.19%
crypt		99	59,039	0.41%
drivers		15,564	8,342,761	57.69%
firmware		259	280	0.00%
fs		14	1,047,859	7.25%
include		1,902	535,531	3.75%
init		3,203	3,384	0.02%
ipc		16	8,291	0.06%
kernel		290	196,039	1.36%
lib		242	51,489	0.36%
mm		82	88,463	0.61%
net		1,584	733,144	5.07%
samples		72	3,268	0.02%
scripts		295	30,985	0.21%
security		204	67,790	0.47%
sound		1,712	742,209	5.13%
tools		710	87,413	0.60%
usr		5	630	0.00%
virt		17	6,480	0.04%
total		47,061	14,462,541	100.00%

[3] ls -R ./DIR/ | wc -l

[4] find ./DIR/ -name *.[ch] -exec wc -l {} | gawk ' n += $1 END print n '

分散並行開発を支えるコード管理システム　= git

誰がリナックスカーネルのコードを書いているのか

Table : Most active 3.5 developers[5]

By changesets			By changed lines		
Greg Kroah-Hartman	239	2.2%	Paul Gortmaker	44,000	5.7%
Axel Lin	191	1.7%	Viresh Kumar	20,425	2.7%
Mark Brown	187	1.7%	Steven Rostedt	14,615	1.9%
H. Hartley Sweeten	135	1.2%	H. Hartley Sweeten	13,083	1.7%
David S. Miller	131	1.2%	Dave Airlie	12,217	1.6%
Daniel Vetter	130	1.2%	Sakari Ailus	10,835	1.4%
Al Viro	128	1.2%	Dong Aisheng	10,574	1.4%
Stephen Warren	121	1.1%	Sonic Zhang	10,494	1.4%
Tejun Heo	112	1.0%	Paul Walmsley	10,084	1.3%
Eric Dumazet	105	1.0%	Ben Skeggs	10,000	1.3%
Hans Verkuil	102	0.9%	Rob Herring	9,886	1.3%
Paul Mundt	102	0.9%	Sascha Hauer	9,602	1.3%
Johannes Berg	102	0.9%	Stephen Warren	9,365	1.2%
Shawn Guo	102	0.9%	Parav Pandit	8,846	1.2%
Thomas Gleixner	98	0.9%	Nicholas Bellinger	8,704	1.1%
Dan Carpenter	86	0.8%	Linus Walleij	8,496	1.1%

[5] http://lwn.net/Articles/507986/

誰がリナックスカーネルのコードを書いているのか　2

Table : Top 3.5 changeset contributors by employer[6]

(None)	1,343	12.3%	Samsung	251	2.3%
Red Hat	1,123	10.2%	Oracle	204	1.9%
Intel	1,061	9.7%	Renesas Electronics	201	1.8%
(Unknown)	860	7.8%	MiTAC	191	1.7%
Linaro	519	4.7%	NVIDIA	188	1.7%
Novell	440	4.0%	Wolfson Microelectronics	187	1.7%
Texas Instruments	313	2.9%	(Consultant)	160	1.5%
IBM	282	2.6%	NetApp	153	1.4%
Linux Foundation	279	2.5%	Vision Engraving Systems	135	1.2%
Google	265	2.4%	Qualcomm	121	1.1%

[6] http://lwn.net/Articles/507986/

カーネル開発の20年の歴史（全体）

This year is 20th anniversary of Linux

- Linux is growing with rapid pace and further acceleration is on going from 2009

柴田さん講演資料

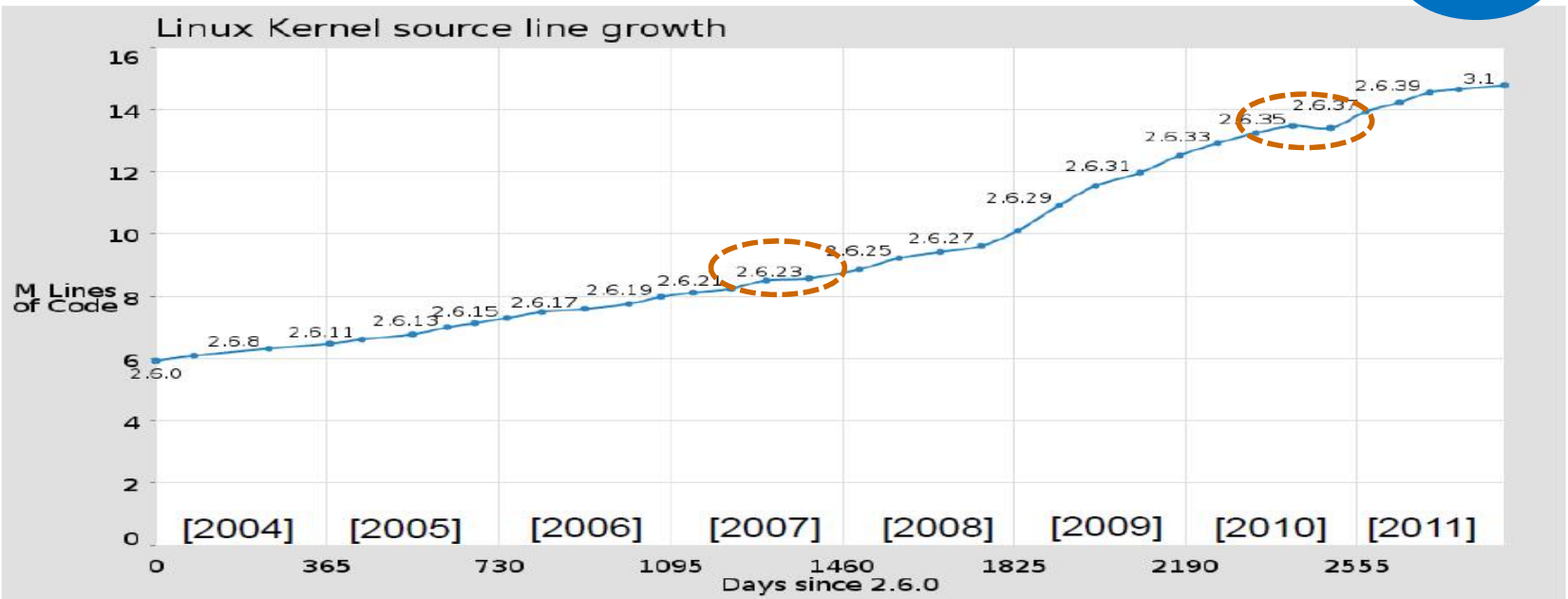
20年の歴史 の中で 開発プロセス革新 により生産性を高めながら成長を続けてきた

カーネル開発の20年の歴史（最近）

Source code growth

- 0.8ML/year from 2004-2008, 1.8ML/year since 2009,
- Large code removal happened 2.6.23 and 2.6.36

柴田さん講演資料



最新の リナックスカーネル 3.1 は *37,085 ファイル、14,770,555 行* の規模

何故デジタル家電業界はリナックスにたどり着いたのか

最新技術が常に今までより良い結果をもたらすとは限らない

- アナログテレビのチャンネル切り替えはずっと早かった！
- フィーチャーフォンの電池はずっと長くもった！
- 昔の電子機器は電源を入れればすぐに使える状態になった！

昔のアプリケーションを同じように動かしたいだけなら、リナックスを使う必要なんて無いですね。何故リナックスを使わなければならなかったかを考えて見ることには重要な意味があります。

こんな事を実現したいならリナックスは必須ですね…

- Twitter, Facebook などのソーシャルアプリ対応の携帯電話
- Youtube コンテンツの再生に対応したテレビやプレイヤー
- クラウドコンピューティングに連携した家電機器

苦難の歴史を振り返る

現在、デジタル家電業界の多くの機器でリナックスが使われるようになっていますが、ここに至るまでには、極めて多くの苦労の歴史の積み重ねがありました。これからリナックスの 活用を考える人は、デジタル家電業界の先人たちと同じ失敗をしたいように過去の苦労の歴史から『これをやったら失敗する』という禁じ手を学ぶべきです。

デジタル家電業界の勘違い

- 経営者はリナックスを商用ソフト安価な代替と考えた。
- 開発者は自分たちがリナックスのユーザーと考えた。
- デジタル家電向けリナックスの要求仕様書を作り、コミュニティにそれに準拠した リナックスを開発するように要請したが、そんなリナックスはリリースされなかった。逆に開発コミュニティから仕様書ではなくパッチを出せと言われた。
- リナックス上で RTOS プログラムを使いたいと思った
- しまいには家電向けリナックスという亜流を作り出してしまった

散々苦労した末にわかったこと

デジタル家電業界のあるべきリナックスの活用法

- バージョンアップ時に膨大な作業が必要になるので、マスターコードから派生した自分流の改造リナックスを作らない。
- あなたの欲しいコードは既に開発済みかもしれないので、コミュニティの開発者の成果も常にウォッチしておく。
- git の活用など コミュニティで実績のある手法を導入する
- (git-hub, ML and Wiki) の参照など公開情報を有効活用する
- できる限りコードを Upstream に投稿する

製品開発現場と開発コミュニティの橋渡し役になるような オープンソース開発専任者を自社内に配置するのはオープンソース活用のための有効な戦略になる。

Linux vs RTOS, 技術的にどうちがうのか

リナックスの本質（技術的な側面）

リナックスの基本的な考え方は heuristics（実行結果の蓄積による類推）による自動調整です。恣意的に自動調停を妨害するようなコードを書いてしまうとシステム全体の動作に悪影響が出るので 自動調停のメカニズムが正しく機能できるような正しいコードを書くべきなのです。

	Linux	RTOS
scheduling	heuristics	deterministic
resource allocation	flexible	fixed
execution order	runtime coordination	predictable
MMU (virtual address)	support by default	not support
multi-processor	support by default	not support
virtualization	yes (KVM, others)	not support
address description	abstracted	absolute value

1. カーネル空間 と ユーザー空間 の明確な分離

完全にデバッグされた RTOS プログラムを Linux に移植する場合

出来るだけオリジナルを継承する

- 間接アドレッシングを回避するために、ユーザー空間から mmap 経由でレジスタにアクセスさせる
- カーネルサービスは非活用

POSIX 界面で明解に分離する

- デバイスドライバー新規開発
- standard POSIX API による明解な空間分離の適用
- マルチスレッドに対応したプログラミングを行う

1. カーネル空間 と ユーザー空間 の明確な分離

完全にデバッグされた RTOS プログラムを Linux に移植する場合

出来るだけオリジナルを継承する

- 間接アドレッシングを回避するために、ユーザー空間から mmap 経由でレジスタにアクセスさせる
- カーネルサービスは非活用

POSIX 界面で明解に分離する

- デバイスドライバー新規開発
- standard POSIX API による明解な空間分離の適用
- マルチスレッドに対応したプログラミングを行う

正しい戦略は…

POSIX による明解な空間分離はリナックスの基本戦略要件です

2. デバイスドライバーの設計

SoC やボード上のデバイス制御用にドライバーを設計する時に

全機能が使えるドライバー

- 最初から コントローラーの全機能に対応する
- 独自の ioctl 拡張 を利用
- メインライン化は考えない

機能限定のドライバー

- リナックス標準フレームワークと互換の API を継承
- コントローラーの新機能が利用できるよう、フレームワーク側の拡張を提案していく
- メインライン化を促進

2. デバイスドライバーの設計

SoC やボード上のデバイス制御用にドライバーを設計する時に

全機能が使えるドライバー

- 最初から コントローラーの全機能に対応する
- 独自の ioctl 拡張 を利用
- メインライン化は考えない

機能限定のドライバー

- リナックス標準フレームワークと互換の API を継承
- コントローラーの新機能が利用できるよう、フレームワーク側の拡張を提案していく
- メインライン化を促進

正しい戦略は…

リナックスの標準フレームワークとの API 互換性の方が重要

3. タスクの実行順序（優先度）のコントロール

応答性を改善するためのカーネルスケジューリングの調整

自動スケジューラを信頼

- レイテンシーの最小化をスケジューラの自動制御で実現
- プロセスの動的優先度自動調整機能を活用
- 積極的に CPU 資源を解放

詳細な手動チューニング

- プロセス毎の nice 値 を詳細に手動設定する
- リアルタイムスケジューリングポリシー の積極的な利用
- udelay (busy wait) によるタイミング調整の多用

3. タスクの実行順序（優先度）のコントロール

応答性を改善するためのカーネルスケジューリングの調整

自動スケジューラを信頼

- レイテンシーの最小化をスケジューラの自動制御で実現
- プロセスの動的優先度自動調整機能を活用
- 積極的に CPU 資源を解放

詳細な手動チューニング

- プロセス毎の nice 値 を詳細に手動設定する
- リアルタイムスケジューリングポリシー の積極的な利用
- udelay (busy wait) によるタイミング調整の多用

正しい戦略は…

過剰なスケジューリングへの介入で CPU（資源）をブロックしない

4. リソース（メモリー資源）のアロケーション

画像バッファー用に大きな物理連続メモリーを確保する場合

スタティックにメモリーを確保

- カーネル管理外の領域に静的に連続領域を確保する
- 特定のアプリケーションだけがその領域を占有する

ダイナミックにメモリーを確保

- カーネル機能でメモリーを割りあて、low-memory 検出機能を活用
- memory pool を活用し断片化を抑止する
- CMA による連続メモリー領域の確保

4. リソース（メモリー資源）のアロケーション

画像バッファー用に大きな物理連続メモリーを確保する場合

スタティックにメモリーを確保

- カーネル管理外の領域に静的に連続領域を確保する
- 特定のアプリケーションだけがその領域を占有する

ダイナミックにメモリーを確保

- カーネル機能でメモリーを割りあて、low-memory 検出機能を活用
- memory pool を活用し断片化を抑止する
- CMA による連続メモリー領域の確保

正しい戦略は…

リナックスの 投機的な動的メモリー割り当て は極めて効率的

5. バイナリーオブジェクト (binary blobs)

リナックスでのバイナリーコード（ライブラリー等）の扱い

バイナリーは出来るだけ回避する

- カーネル空間はGPLライセンス (ソースコード入手は担保)
- ユーザー空間ライブラリーのソースも出来るだけ確保する
- シンボル情報はデバッグ時には必須である

バイナリーオブジェクトを許す

- ベンダーがバイナリー形式でミドルウエア等が提供する
- ベース環境 (kernel, toolchain) の更新が必要になった場合には ソースが無いとリビルドが出来ず非常に困ることになる

5. バイナリーオブジェクト (binary blobs)

リナックスでのバイナリーコード（ライブラリー等）の扱い

バイナリーは出来るだけ回避する

- カーネル空間はGPLライセンス (ソースコード入手は担保)
- ユーザー空間ライブラリーのソースも出来るだけ確保する
- シンボル情報はデバッグ時には必須である

バイナリーオブジェクトを許す

- ベンダーがバイナリー形式でミドルウエア等が提供する
- ベース環境 (kernel, toolchain) の更新が必要になった場合には ソースが無いとリビルドが出来ず非常に困ることになる

正しい戦略は…

問題を回避するために出来るだけバイナリーオブジェクトを回避

6. リナックスの開発プロセスとツール

パッチレビュー / インテグレーション / バグトラッキング

自社内プロセスの継続利用

- 実績のあるツール
- メンテナーによるレビューがないまま結合、その後テスト&デバッグ
- 不適切な git の利用 (too large commit, no log)

リナックス標準ツールの導入

- 結合前に全てのコードの妥当性を厳しく評価する 社内メンテナー を配置する
- git を正しく運用する
- 公知情報（コード、議論、バグ情報　等）を活用する

6. リナックスの開発プロセスとツール

パッチレビュー / インテグレーション / バグトラッキング

自社内プロセスの継続利用

- 実績のあるツール
- メンテナーによるレビューがないまま結合、その後テスト&デバッグ
- 不適切な git の利用 (too large commit, no log)

リナックス標準ツールの導入

- 結合前に全てのコードの妥当性を厳しく評価する 社内メンテナー を配置する
- git を正しく運用する
- 公知情報（コード、議論、バグ情報　等）を活用する

正しい戦略は…

リナックス開発で実証された開発手法を導入し積極活用するべき

7. カーネルバージョンの選定基準（LTS / LTSI）

バージョンによってメンテナンス期間が異なる点に注意

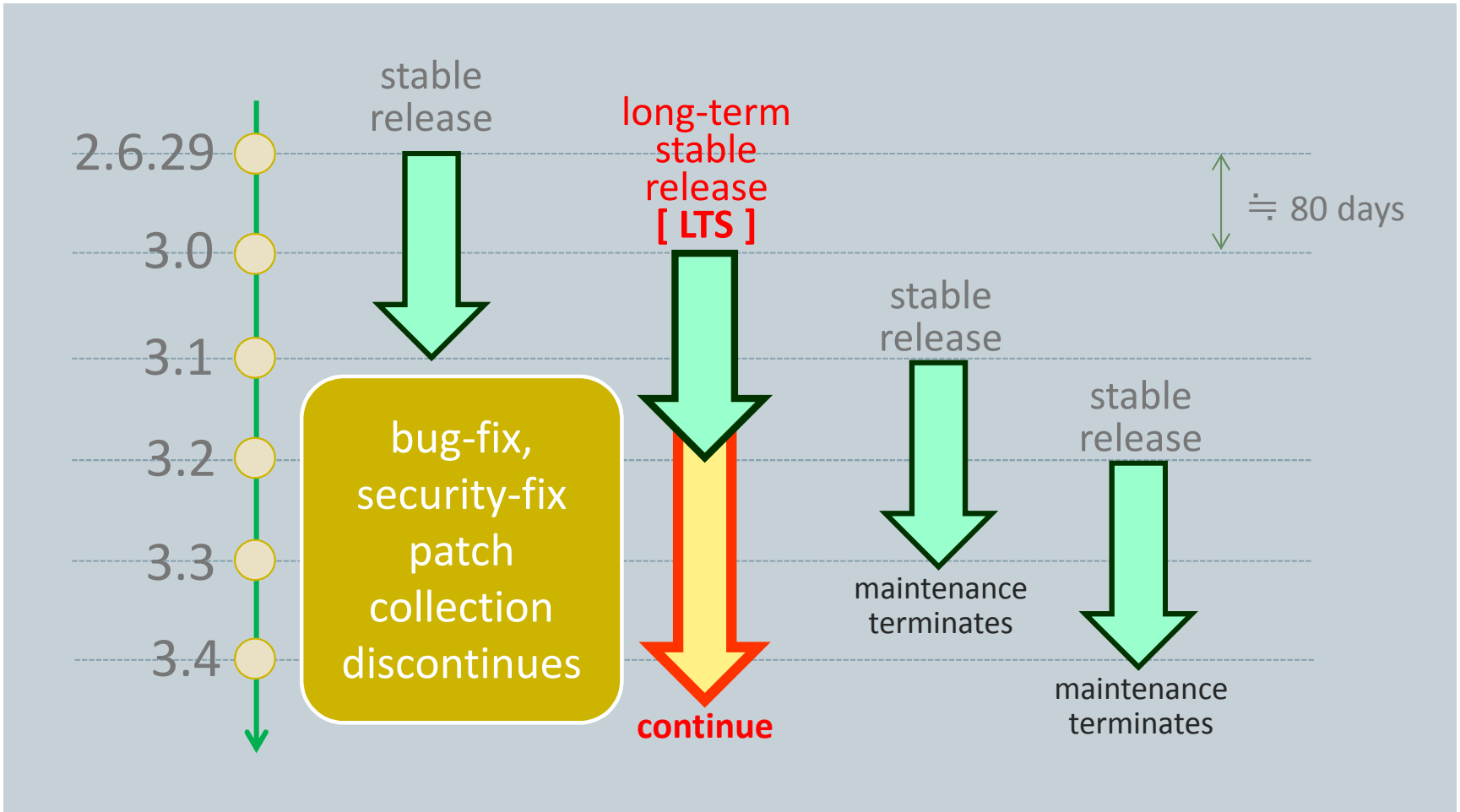
単純に貰ったカーネルを利用

- SoC ベンダーの BSP のカーネルをそのまま利用
- ディストリビューションやプラットフォーム（Android, Genivi 等）を利用
- カーネルは変更したくない

意識的にバージョンを選択

- 常に最新を使いたい
- コミュニティの LTS
- LF/CEWG LTSI (Long-Term Stable kernel for Industry)

開発コミュニティのカーネルメンテナンススキーム



バージョン 3.4 が次期 LTS (と LTSI) バージョン

https://lkml.org/lkml/2012/8/20/675 より

the 3.4 kernel tree will be -longterm

From: Greg KH
Date: Mon Aug 20 2012 - 18:25:09 EST

- Next message: Andrew Morton: "Re: [PATCH v3 3/9] rbtree: place easiest case first in rb_erase()"
- Previous message: Shirley Ma: "Re: [RFC PATCH 1/1] fair.c: Add/Export find_idlest_perfer_cpu API"
- Messages sorted by: [date] [thread] [subject] [author]

As I'm getting a few questions about this, and I realized that I never sent out an email about this, yes, the 3.4 kernel tree will be the next -longterm kernel that I will be maintaining for at least 2 years.

Currently I'm maintaining the following stable kernel trees for the following amount of time:
3.0 - for at least one more year
3.4 - for at least two years
3.5 - until 3.6.1 is out

Hope this helps clear up any rumors floating around. If anyone has any questions, please let me know.

greg k-h

7. カーネルバージョンの選定基準（LTS / LTSI）

バージョンによってメンテナンス期間が異なる点に注意

単純に貰ったカーネルを利用

- SoC ベンダーの BSP のカーネルをそのまま利用
- ディストリビューションやプラットフォーム（Android, Genivi 等）を利用
- カーネルは変更したくない

意識的にバージョンを選択

- 常に最新を使いたい
- コミュニティの LTS
- LF/CEWG LTSI (Long-Term Stable kernel for Industry)

正しい戦略は…

LTSI ベースの長期安定版を採用する（させる）べきである

8. コミュニティへの投稿　アップストリーミング

何故アップストリーム？　どんな価値があるの？

利用するだけで充分では？

- 単なる受身ユーザーだから
- 積極的なパッチ投稿の機会／動機がない
- 必要があれば自社内でカーネル改造できるから
- 社内ツリーを管理する

積極的に取り組みたい！

- 積極的にフィードバック
- アップストリームで対応済みのデバイスを使う
- アップストリームとの差分を最小化する戦略

8. コミュニティへの投稿　アップストリーミング

何故アップストリーム？　どんな価値があるの？

利用するだけで充分では？

- 単なる受身ユーザーだから
- 積極的なパッチ投稿の機会／動機がない
- 必要があれば自社内でカーネル改造できるから
- 社内ツリーを管理する

積極的に取り組みたい！

- 積極的にフィードバック
- アップストリームで対応済みのデバイスを使う
- アップストリームとの差分を最小化する戦略

正しい戦略は…

企業からのパッチ投稿が非常に増えている意味を考えるべき

9. リナックスディストリビューション

パッケージ(=アプリケーション)間の互換性を検証する

自分で整合性を確保できる

- 単純なヘッドレス機器
- OSS ミドルウエアなし
- OSS ライブラリなし
- OSS アプリなし
- ソースも自己管理

ディストリビューションを活用

- PC と同じようなパッケージ管理を利用したい
- 包括的にバイナリー互換性を確保したい
- 製品出荷後もセキュリティ対策パッチの提供が必要

9. リナックスディストリビューション

パッケージ(=アプリケーション)間の互換性を検証する

自分で整合性を確保できる

- 単純なヘッドレス機器
- OSS ミドルウエアなし
- OSS ライブラリなし
- OSS アプリなし
- ソースも自己管理

ディストリビューションを活用

- PC と同じようなパッケージ管理を利用したい
- 包括的にバイナリー互換性を確保したい
- 製品出荷後もセキュリティ対策パッチの提供が必要

正しい戦略は…

大規模システムでは組み込みでもディストロ活用が必要

市場シェアNo. 1獲得は最大のリスク要因になる

従来デジタル家電分野で世界市場を席巻してきた日本企業は、新興国から登場したコンペチターとの熾烈な競争環境の中でこれまで正常進化型（連続的）のイノベーションを積み重ねているが、ある日全く想定外の競争環境の変化が発生し急速に競争条件が変化してしまうリスクを持っていることを認識しておく必要がある

イノベータのジレンマ [Clayton Christensen]

- 現在技術の延長上ではないところから新しい技術が登場し、唐突に主役を奪ってしまう（破壊的イノベーション）
- 現状で最適なオペレーションが命取り
- 今日の「勝ち組」が明日の「負け組」になる

先端競争分野では急速な寡占化が進行していて、日本企業はスマートフォン等からの撤退を余儀なくされた。今後他のデジタル家電製品がソリューション指向になれば同様の事態が発生する可能性が高い。

持続的イノベーション

持続的イノベーション (Sustaining Innovation) の特性

- 今使うのに最適な技術の改善
- 今までよりすぐれた性能を提供
- 現在マーケットでシェアや経験をもっている大企業が有利
- この戦略の問題点はイノベーションのジレンマの誘引

イノベーションのジレンマ誘引の典型例 (= 日本産業の苦悩)

- 優良顧客の声だけを聞いて、新しい消費者にフォーカスしそこなう(新参者に未開拓市場を奪われる)
- 過剰品質・機能・性能を追求しているうちに、別の観点からの競争に出遅れてしまう

破壊的イノベーション

破壊的イノベーション (Disruptive Innovation) の特性

- これから伸びてくる技術
- 市場を塗り替えてしまう技術革新
- 別の次元で勝負、新しい市場の創造
- 新しいユーザーの獲得
- 異分野からの新規参入企業やベンチャー企業が有利
  (「しがらみ」がない、既成概念に囚われない)

我々は破壊的イノベーターが参入して市場が破壊されてしまうより前に、過去の成功体験を捨てて(あえて否定をして)自らが改革者とならねば生き延びられないことを認識する必要がある。過去の単純な延長に未来は無く革新なくしては生き延びられないのだが 同時に、革新は努力の積み重ねの結果達成されるものであり、突然変異的に得られるものではない事を理解しておこう。

出た当初に破壊的な革新に気づく人は少ない...

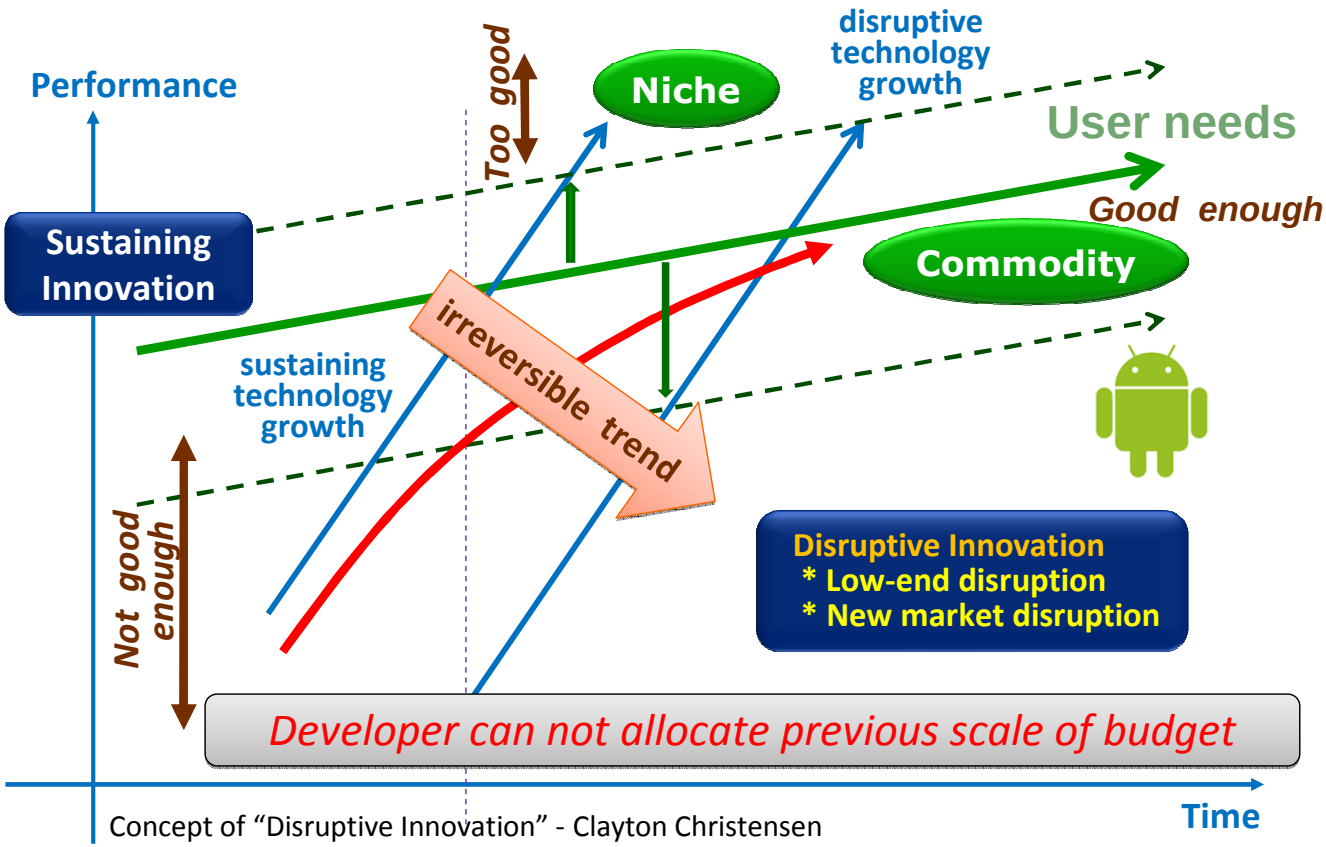
破壊的イノベーション過去事例

- マイクロプロセサ (電卓の部品、1971)
- PC (8bit CPUのマニア向けゲーム機、1975?)
- ケータイ (ショルダーフォン、3kg、2万円/月、1985)
- 液晶TV (1.2型、1982、3インチ、9万画素、1987)
- MP3 (CDより劣った音質、1991)
- Linux (大学生の学習用OS、1991)
- デジカメ (25万画素、1.8型液晶、1995)

1990年当時組み込み機器でLinuxを動作させるのは一部のマニアの趣味と考えられていた。半導体会社でLinuxをビジネス商材と考えた人はいなかったが、感性に優れたPC周辺機器メーカーの幹部に発掘される形で家庭用のNASという新ジャンル製品が誕生

破壊的イノベーションは不可逆的である

Commoditization and “low-end disruption”



Concept of “Disruptive Innovation” - Clayton Christensen

DEVCON Solutions for Innovation
RENESAS

日本のデジタル産業界が直面する課題

課題

- 国内産業の空洞化（お得意様が負け組ばかりに）
- 結果として国内販売体制の余剰感、一方で海外が極めて手薄
- 国内の売り方（＝手厚い対面サポート）が通用しない
- 製品ラインナップの混乱（製品多すぎ）
- 製品の背後には専属の設計部隊を沢山抱える（非効率）
- システム技術、ソフトウエア技術への対応が困難
- オープンプラットフォームなど新潮流への対応
- 海外におけるブランドの弱さ
- 国際コミュニケーション力、意思決定のスピード感の欠如

現在日本の電子産業が直面するリスク

リスク

- デバイス自体は本質的に交換可能で過当競争に陥りやすい
- デバイスの高度化に伴うサポートコスト増大、収益性の低下
- 国内においては黒船の到来、結果としての競争条件の変化
- ジャパンプレミアム前提の高コスト体質で国際競争力なし
- 破壊イノベーションによる急激な競争環境の変化
- ソフトウエア技術力の国際競争力の弱さ（ガラパゴス化）
- コア技術を社外に依存した単なる手配師は中抜きされる

感度をあげて新しい脅威を感知する(1)

Tim O'reillyの有望技術発見基準

- hackability
  - その道のプロをにやりとさせるものがある
- being in line with some major trend
  - 追い風が吹いている
- disruptive potential
  - なにかをぶっとばしかねない迫力がある
- grassroots enthusiasm
  - 熱気にみちたひとびとの支持がある
- the presence of professional practitioners
  - 仕事をきちんとこなすプロがいる
- a possible business ecology
  - 商売になりそう、ビジネス生態系が成立しうる

コラム　人間は実際には驚くほど現実を見ていない

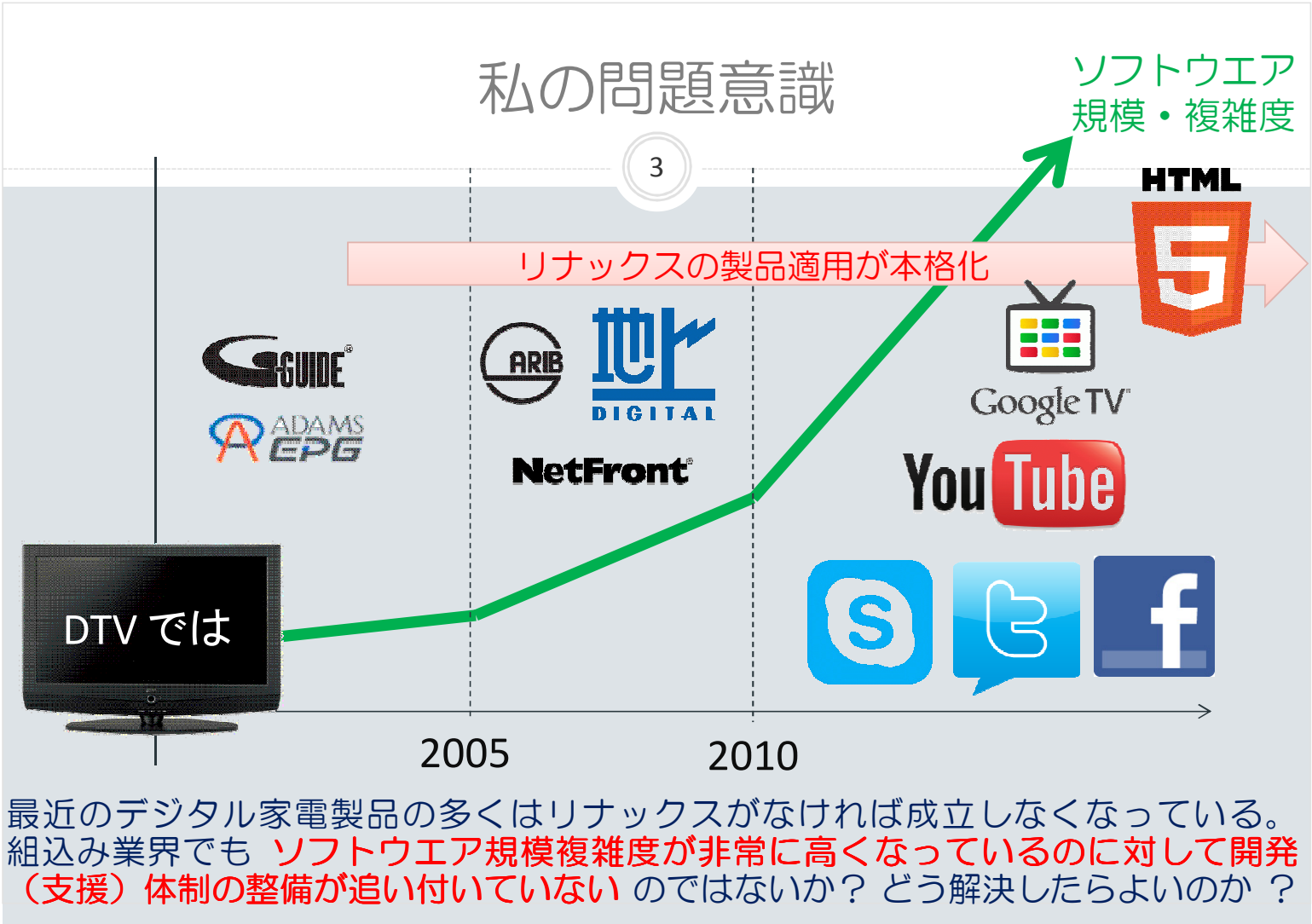
こんちには みさなん おんげき ですか？　わしたは げんき です。
この ぶんょしう は いりぎす の ケブンッリジ だがいく の けゅきんう の けっか
にんんげは もじ を にしんき する とき その さしいょ と さいご の もさじえ あいてっれば
じばんゅん は めくちちゃゃ でも ちんゃと よめる という けゅきんう に もづいとて
わざと もじの じんばゅん を いかれえて あまりす。
どでうす？　ちんゃと よゃちめう でしょ？
ちんゃと よためら はのんう よしろく

みさなん ちんゃと よまめした か？
にんんげ には とても ふぎしな こと が たさくん あまりす ね。
しらばく このうよな ふぎしな ぶんょしう で にっき を かこうかしら？
でも この ぶんょしう を かがんえて いると にもじ や さもんじ の
いかれえ が でなきい ことば ばかり が おもい うかぶ f-^;
けこっう むかずしい です。

完全なる商品("the Whole Product") という概念

- ユーザが望むものは裸のハードウェアだけではない
  - ソフトウェア(OS、アプリケーション、...)
  - 周辺装置、アクセサリ
    (ストレージ、ネットワーク、ケーブル、カメラ、...)
  - ネットワーク・インフラ、サービス、コンテンツ
  - 開発環境、トレーニング、サポート
  - システム・インテグレーション
- 評価用サンプルソフトだけではソリューションにはならない
- バリューチェイン(価値連鎖) の提供
  - 本当の意味でのソリューション提供は「困っている管理者」の問題を解決してあげることであるから、提供を期待されるのは単なる技術供与だけに止まらない
  - もはや1社ですべてを提供することは不可能 → 協業による課題解決提案力も必要

我々が解決すべきユーザーの本質的課題は何か?….



最近のデジタル家電製品の多くはリナックスがなければ成立しなくなっている。
組込み業界でも ソフトウエア規模複雑度が非常に高くなっているのに対して開発（支援）体制の整備が追い付いていない のではないか？ どう解決したらよいのか ？

日本企業に残されたオポチュニティ

本質的には技術の複雑度があがればあがる程、ユーザーは自分だけでは問題を解決できなくなる。そのギャップを誰が埋めること（目先の課題解決）を提供するだけではなく、本質的な問題解決となる代案（= ソリューション）を提供する機会が与えられていることに気づくべき。

オポチュニティ

- ユーザーは”メーカー以上” を期待されている
- 製品を仕上げきる力（顧客との完全利害一致）
- 技術革新による競争条件の変化（もはやPCの時代ではない）
- 技術サポートにおいても ”おもてなし” の対価性

感度をあげて新しい脅威を感知する(2)

破壊的イノベーションの芽をみつけるためのキーワード

- 遅くて / ややこしくて / 重すぎて / 高すぎて / 難しすぎて、使いものにならない
- 商品のクォリティでない
- 安物、まがいもの
- おもちゃ趣味で使うならいいけど。マニア向け
- きれいごと。世間じゃ通用しない。わりきりすぎ

今目の前で起こっている先駆けとなる兆候をきちんと正視しなければならない。「あれは特殊ケースだから……」で片付けてしまったらせっかくのチャンスが台無しである。むしろ今は荒唐無稽に見えるような発想に着目すべきである。

まとめ

- オープンソースソフトウエアの代表例として、リナックスカーネルの概要とその開発プロセスについて紹介した。今後ソフトウエア技術者として活躍して いく上ではオープンソースを活用していくだけでなく、オープンソース開発者との連携がポイントになると認識して欲しい。
- 日本のデジタル家電産業が従来のRTOSからリナックスを導入した時に経験した失敗例を参考にしながら、組み込み機器の開発にリナックスを導入する際の ヒントとなる事項を9件紹介した。リナックスのような大規模ソフトウエアを利用する場合には適切な作法を理解して正しい実装をする事が重要である。
- クリステンセンの『破壊的イノベーション』の概念を参考にしながら、日本の電子産業が直面するイノベーションのジレンマについて検討した。従来の 延長線での改善（＝継続的イノベーション）だけでは生き残れないので柔らかい頭で次世代に通じる飛躍を目指してほしい。