

卒業論文 2002 年度（平成 14 年度）

デバイス・ハブ構想

慶應義塾大学 総合政策学部 4 年

久松 慎一

学籍番号 79907360

指導教員

村井純

徳田英幸

楠本博之

中村修

南政樹

平成 15 年 1 月 21 日

概要

マイクロコンピュータの高性能化によって、それらが単体でネットワーク端末として機能し始めた。

ネット家電や NAS (Network Attached Storage) といった低コスト/省スペース/低消費電力といったマイクロコンピュータの特性を活かした新しい提案がどんどん市場にでてきている。

これらの多くは既存の(枯れた) ネットワークテクノロジーとを使用しつつ、新しいアプリケーションを生み出している。

そこで、本研究では USB や IDE といったパーソナルコンピュータのローカルポート/バスをネットワーク上に展開するデバイス・ハブ構想の提案、その環境のグランドデザインを行う。デバイス・ハブにより、ネットワーク上で自由にホストとデバイスを組み合わせてコンピュータ環境を構築出来る。その際 OS にはデバイス・ハブ デバイスであることを認識させることない。つまりドライバをインストールするなどの特別な変更が必要ない。

また、実際にデバイス・ハブ環境を構築し、またその対応デバイスを実装することでその実現可能性と有用性を実証し、将来への諸検討を行った。

キーワード: コンピュータネットワーク デバイス

Abstract

Microcomputer with high-performance have been advanced and it itself started to function as an apparatus(a device) for network terminal.

In these days new proposals concerning Internet appliance and NAS(Network Attached Storage), that take advantage of microcomputer's characteristics such as low-cost, small-footprint and a little electrical consumption are remarkable.

And, (more and) more of there proposals are offered in the market.

Although most of these use legacy system in the context of network technologies, they are generating new applications.

Thus, this paper presents a grand design for "Device-Hub concept", that is the idea to connect personal computers' local ports/ buses such as USB or IDE to network.

Within the Device-Hub concept, personal computer as a host and devices attached the host are supported in network environment while the users are not bothered about set-up. Moreover this environment does not require them to install drivers, thus Operating System does not need to recognize devices attached as a part of "Device-Hub" devices.

In this paper "Device-Hub" system was developed. Furthermore, with the implementation of devices that corresponding to this system, its feasibility and usability are made certain. In conclusion, evaluation based on the implementation and views towards future use are mentioned.

Keyword: computernetwork device

目次

目次	4
図一覧表.....	7
表一覧表.....	8
1 序章.....	9
1-1 背景	9
1-2 目的	10
1-3 本論文の構成	11
2 デバイス・ハブ構想	12
2-1 コンセプト.....	12
2-2 機能要件.....	15
2-3 デバイス・ハブの特徴.....	16
2-3-1 レガシーポート・バス	16
2-3-2 ZeroConf	16
3 類似事例.....	17
3-1 ネット家電.....	17
3-2 NAS	18
3-3 iSCSI	19
3-4 Screen	20
4 デバイス・ハブ環境の設計	21
4-1 ネットワーク上のモデル図	21
4-1-1 Hybrid P2P モデル	21
4-1-2 ZeroConf モデル	22
4-3 マッチング部の設計	23
4-3-1 Hybrid-P2P モデル	23
4-3-2 ZeroConf モデル	23
4-3-3 クライアント	25

4-4-1 IPv4 の場合.....	25
4-4-2 IPv6 の場合.....	25
4-4 ホストデバイス間通信	27
5 実装	28
5-1 USB のネットワーク化について	28
5-1-1 特徴.....	28
5-1-2 USB と Firewire	29
5-1-3 トポロジ.....	29
5-1-4 データ転送タイプ	31
5-1-5 バス速度.....	32
5-1-6 エンドポイント.....	32
5-1-7 インタフェース.....	33
5-1-8 プロトコルデザイン.....	34
5-1-9 パケット.....	34
5-1-10 トランザクション	35
5-1-11 クラス.....	35
5-2 USB キーボードの実装.....	37
5-2-1 USB プロトコルについて	37
5-3 インタフェースノード実装について	39
5-3-1 mozuku について	39
5-3-2 TRON について.....	39
5-4 インタフェースノード実装環境.....	44
5-4-1 H8 について.....	44
5-4-2 RTL について.....	46
5-4-3 USBN9603 について.....	46
5-4-4 クロスコンパイル環境	47
5-5 実装項目	49
5-5-1 サーバ	49
5-5-2 クライアント	50
5-6 実装上のポイント	51

5-7 完成品	52
6 評価	54
6-1 評価環境について.....	54
7 デバイス・ハブを展開するには.....	57
7-1 対応バス・ポート	57
7-2 マッチングポリシー	58
7-2-1 空間をキーとしたマッチング	58
7-2-2 所有者をキーとしたマッチング.....	59
7-3 要求事項	60
7-3-1 セキュリティ	60
7-3-2 帯域制御.....	60
8 まとめ	61

図一覧表

図 3-1	I/O data 社製 i-connect - LAN コンバータ LAN-iCN	18
図 4-1	ハイブリッド P2P モデル図	21
図 4-2	ZeroConf モデル図	22
図 4-3	デバイス-ホスト間通信	27
図 5-1	実装システム概要図	28
図 5-2	USB 物理的トポロジ	30
図 5-3	USB 論理的トポロジ	30
図 5-4	単インタフェースを持つ USB プリンタデバイス構成	33
図 5-5	複数のインタフェースを持つ USB プリンタデバイス構成	34
図 5-6	USB スタック図	37
図 5-7	TRON HDFS	40
図 5-8	H8-3069 ブロック図	46
図 5-9	インタフェースノード(ホスト側)	52
図 5-10	USB インタフェース部	53
図 6-1	実験トポロジ図	54
図 6-2	実験トポロジ図	55

表一覧表

表 5-1 転送モードごとに使用可能な転送タイプ	32
--------------------------	----

1 序章

1-1 背景

現在マイクロコンピュータ(以下マイコン)はいろいろな機器に組み込まれている。特に 8bit、16bit レベルのマイコンは分野を問わず多くの製品に使用されている。

開発環境等もふくめて個人で入手可能なマイコンも高速/高性能化が著しい。Z80[1]から PIC[2]へ、そして AVR[3]へと言う流れは自身の経験としても印象深い。そして現在、H8 や Super-H、ARM といった CPU、また CPLD や FPGA レベルのロジックデバイスまでも個人が入手し、遊べるレベルにまできた。

前述の CPU はシステムの構築に必要なメモリや A/D・D/A コンバータ、シリアル通信など多くの周辺機能を内蔵しており、コンパクトで低消費電力なシステムを組むことができる。また、外部の RAM センサ・インタフェースデバイスを組み合わせることで多様な機能拡張が可能である。ネットワークインタフェースを組み合わせることによってマイコン自身がネットワークの端末として動作することが可能である。また TINI[4]の様にそれに特化した特色有るパッケージも登場している。

ロボットコンテストへの参加以来マイコンに憧れを持ち続けてきた著者にとってこの状況は非常に興味深い。

マイコンがネットワーク端末の役割を担うことにより、ネットワーク家電をはじめとしてネットワークの新しい使い方が提案されてきている。これらの多くは機器のコントロールや状態のセンサリングなど、ネットワークテクノロジーとしては特段新しいものではない。しかしながらこれらの提案は新しい産業・生活をリ・デザインしつつある。今後、多くの機器が「ネットワーク対応」を売りにして来ると予想される。

しかしながらそれらの多くがそれぞれの、またベンダーごとにプロトコル・ポリシー設計されており、互換性に乏しく、また複雑化している。またそれぞれに専用のプログラムが必要であるなど、使い勝手もよくない。

1-2 目的

本論文では PS/2,USB,DVI や IDE といったローカルポート、ローカルバスをネットワーク上に展開することによってコンピュータとデバイスをマッチングするプラットフォームについて検討する。このことにより、ホストや各デバイスは物理的な位置等の制約から解放され、また複数のホストで共有することが可能となる。

さらにそれらを家電や AV 機器、OA・FA 機材と言ったコンピュータのデバイス以外へと応用させ、それぞれのデバイスが独立して個々のネットワーク端末となり、それらを適宜マッチングしていくことにより柔軟に周辺機器を組み合わせしていく「デバイス・ハブ」環境を提案したい。

デバイス・ハブ環境においてはネットワーク上の端末がホスト - デバイスという主従関係をもって適宜接続される。例えばユーザインタフェースを搭載しない、または poor な機器に必要な時点でユーザインタフェースデバイスを提供する、また既存のリソースを最大限活用した様々な機器同士のデータ共有/コミュニケーションによる協調・協働環境を提供する。

1-3 本論文の構成

本論文の構成は以下の通りである

1 章「序章」において本研究の背景や目的を記す。

2 章において本研究によって提案される「デバイス・ハブ構想」を解説し、特徴や機能用件事項などをまとめる。

3 章において本研究の類似事例を紹介し、それぞれに対して「デバイス・ハブ」との比較を行う。

4 章においてデバイス・ハブ環境の設計を行う。デバイス・ハブの機能は大きくマッチングとホストデバイス間の通信に分けられる。それぞれに対して用いる技術を整理する。

5 章において、4 章で設計したデバイス・ハブ環境とその下で動作するデバイス・ハブ デバイスの例として USB キーボードのデバイス・ハブ デバイス化を行う。

6 章において、4 章で設計したデバイス・ハブ環境を構築し、5 章の実装を評価し、実現可能性を検証する。

7 章でデバイス・ハブの今後の課題について検討し、8 章で本研究のまとめを行う。

2 デバイス・ハブ構想

デバイス・ハブ構想の概要について以下に述べる。

2-1 コンセプト

現在、パーソナルコンピュータ(以下 PC)にはキーボードやマウス・ディスプレイ・スピーカなどのヒューマンインタフェースやプリンタ・デジタルカメラなどの周辺機器が接続されている。それら周辺機器のポート/バスとしては USB や firewire が普及している。これらは多くのオペレーティングシステム(以下 OS)で標準的にサポートされ、標準的なデバイスで有ればドライバなどによる追加機能不要でホットプラグして使用できる。また、それぞれ標準的なデバイスでなくとも USB/firewire といったポートへのアクセスは多くの OS においてドライバを作成するためのフレームワークが用意されており、それを用いることによってそれらのバスの特性を深く知らなくても容易に周辺機器のドライバを作成する環境が整っている。特に USB はデバイスやデータ転送のタイプによってクラス分けされており、それらを踏襲する形でデバイスドライバを設計することができる。

オフィスユースではプリンタのネットワーク端末化については早くから実用化されている。オフィスユースのプリンタの多くは Ethernet インタフェースを内蔵し、ラインプリンタサーバーや SMB 上のプリンタサーバとしての機能を有する。

同様に、先に列挙したその他の周辺機器類もネットワーク上での端末として扱えないだろうか。現在これらの周辺機器類は PC 本体に直接接続して用いる必要があるため PC 本体と物理的に同じ場所になくなくてはならない。また、複数台の PC でそれらを用いようとする場合はそのたびごとにコネクタを抜き差ししなくてはならないので、手間である。ネットワーク端末化することによってこれらの問題の解決、つまり遠隔アクセスや共有を可能とする。

具体例

例えば、キーボードやマウスなどのヒューマンインタフェースデバイスについて考える。多様な作業内容や携帯性への要求がために複数の PC を並列的に扱う場面は少なくない。その場合、結局のところ同時に作業するのは 1 台の PC であり、ユーザは PC 間を渡り歩くことになる。つまり、ヒューマンインタフェースはユーザとなる人間に対して属されるべきであり、PC ごとに一そろえつつ用意するのは無駄である。このため現在 CPU スイッチャを利用して複数のマシンでヒューマンインタフェースデバイス(主にキーボード、マウス、ディスプレイのセット)を共有し、切り替えて用いることが多く行われている。

またラックマウント型コンピュータなどもそれぞれにヒューマンインタフェースデバイスをもたない。オペレーションの際は

- (1)先ほどと同様 CPU スイッチャを用いる方法
- (2)シリアル経由でターミナルを操作する方法
- (3)telnet や VNC、Remote desktop 等の仮想コンソール環境(ないしそれに準ずるもの)を用いて遠隔操作する方法

などによりアクセスする。

これらに対し、USB や PS/2・DVI をネットワーク上に展開することにより、特殊なソフトウェアを用いることなく PC とヒューマンインタフェースデバイスの間をネットワークで接続し、複数のコンピュータでデバイスをスイッチングして共有すること、さらに OS の標準的な機能のみで遠隔的なオペレーションが可能となる。

ファイルの共有のためにファイルサーバを設置することが行われているが、ハードディスクにマイコンでネットワークインタフェースを追加することにより、ファイルサーバの役割を担うことが出来る。こちらもやはり OS からは IDE アクセスのように見え、ローカルバスへのアクセスと区別されない。HDD 側の通信の多重化はマイコンが実現する。このようにハードディスクへマイコンを追加することで従来必要であったファイルサーバの機能が不要になる。現状、デバイス・ハブ デバイス化によるオーバーヘッドなども勘案すると実用は難しいがネットワークの広帯域かとマイコンの高性能化でこの問題は緩和してゆき、転送速度などの性能が高いレベルで求められない場面において実用的になってゆくもの考える。

一方、後述するようにネットワークに対応した家電や AV 機器なども発売され始めており、今後の普及が見込まれている。これらは、Ethernet のインタフェースを備え、家庭内 LAN に接続することができる。それらの多くは HTTP サーバを内部に持ち、他端末の web ブラウザ上からそれぞれの機器の状態の監視や制御が出来るようになっている。

デバイス・ハブ環境において、これらはホストデバイスという関係をもち論理的に接続されていく。ホスト側が対応している限りデバイスは論理的接続されることでそのホストのリソースの一部となる。

具体例

手元のキーボードやリモコンを適宜切り替えることによってさまざまな機器のヒューマンイ

インタフェースデバイスになる。たとえば家電品においてオペレーションのためのヒューマンインタフェースデバイスは(当然のことながら)製品自体に埋め込まれており、また用いられるのは極短い時間である。デバイス・ハブでヒューマンインタフェースデバイスを接続することによってネットワークへの到達性の範囲で自由にホストを切り替えて操作することが可能となる。プリンタや携帯電話やポータブルプレイヤーのようにスペースやその他の理由でヒューマンインタフェースが貧弱であるかない実装されていない場合にはこのようにその部分を外部化することで条件が許すときのみでもよりリッチなインタフェース環境を享受すること、ないし他と共有しヒューマンインタフェースが必要なときのみそのコンピュータのリソースとして用いることが出来る。

そのような発展がなされると、当然それに従い各デバイスの形式も変化していくことになる。例えばキーボードが家電や AV 機器などのオペレーションに使われるようになるためには LCD の様な小さなディスプレイの役割も含有するようになるなどの変化が見られるであろう。家電のユーザインタフェースを集約するような新しいデバイスが生まれてくることも考えられる。ビジネスユースでも、現在 PDA に専用のソフトウェアを搭載して機器のインタフェースとして用いることが多くなされているが、レガシーポート/バスを利用してヒューマンインタフェースデバイスによるオペレーションを可能とする。

また、デバイス・ハブ環境の場に各自がデバイスを持ち寄ることでホスト側のアプリケーションを共同利用することが出来る。これにより例えばマルチプレイヤーのゲームを実現できる。なにかしらの共通点を持つ者どうしをグルーピングし、そのグループがゲームに参加する。共通点としては物理的に場を共有する、サークルの仲間など多種考えられる。各人のゲームデータやそれにヒモ付けされた ID(多くの場合は IP アドレスとなる)をデバイスに持たせることで、面倒な設定をせずアドホックにゲームを展開できる。また触覚マウス[12]などのタンジブルデバイスをもちいることで空間意識のみならず触覚をもまきこんだ TUI(タンジブルユーザインタフェース)ゲームを展開することが出来る。

2-2 機能要件

デバイス・ハブ環境の実現のために、以下のような機能を必要とする

- ・ ホスト/デバイスのネットワークへの接続

当然ながらデバイス・ハブ環境においてはホスト/デバイスはネットワークに接続されていなければならない。そのためにネットワークコンフィギュレーションの自動化などの機構を考慮する。

- ・ マッチング

適切なポリシーに基づいてホストとデバイスをマッチングする必要がある。

マッチングのポリシーは状況に応じて様々に変化するので柔軟に設計できる必要がある。 また可能な場合はポリシーの追加・削除の機構を用意する。

- ・ ホストーデバイス間の通信

マッチングされたホストとデバイス互いにネゴシエートし、通信を行う。

2-3 デバイス・ハブの特徴

2-3-1 レガシーポート・バス

デバイスとネットワークの間・またネットワークポートとPCとの間にマイコンを噛ませ、ネットワーク化による様々な変更点をマイコンから吸収させることによりデバイスからもOSからもごく普通のポートを介した通信と同様に処理することが出来る。

よってコンピュータ側にもデバイス側にも特別なプロトコルやドライバなどをインストールする必要はなく、コネクタ部分にマイコンを追加することで普通のデバイスをデバイス・ハブ・デバイス化することが出来る。

2-3-2 ZeroConf

ネットワーク的な PLUG-IN を実現するために、ZeroConf 技術[6]の標準化が IETF にて行われている。これは Addressing, Naming, Browsing にそれぞれ address autoconfiguration、multicast DNS、DNS service discovery といった既存の技術を統括的に扱うことでネットワーク接続できるコンピュータなどの電子機器を既存のネットワークに簡単に追加したり複数の機器を手動の接続設定なく即席でネットワーク化することを目指すものである。

address autoconfiguration は IPv4 においては AUTO-IP[20][21]を用いて、IPv6 においては NDP 等を用いてノードが自身のインタフェースのアドレッシングを行う。これにより DHCP サーバが無くとも(stateless な)IP アドレスを自動生成し、ネットワークへの接続性を確保する。

multicastDNS[7][8]は各ノードへの Naming を提供する。domainname や service type・protocol 等をもとにノードへのネーミングを行う。階層的な Naming をすることでコンピュータシステム・ユーザともに親和性の高い Naming が実現される。

DNS service Discovery[9]は multicastDNS によって付けられた名前を元に必要なサービスを lookup し、vender と user をマッチングさせていくものである。

これらは実装的にもネットワーク帯域的にも軽量に設計されており、マイクロデバイスにも適している。現在 Chattiness の問題があるが、これについても議論が続けられており、caching や query/announce の back off 等の方法による解決が試みられている。また、mDNS や DNS-SD のサーバ側実装はオープンソースソフトウェアが公開されており、容易に構築することが出来る。

3 類似事例

デバイス・ハブ構想に類似した先行研究/事例について3つ挙げ、紹介する。

3-1 ネット家電

現在、家電が Ethernet へ接続するインタフェースを備えつつある。家電がネットワークにつながることで、その家電本来の機能にくわえさらに発展した使い方が生まれてくる。冷蔵庫やエアコンを散った家電を単体で使うだけでなく、それらをまとめて操作することが考えられているおり、レシピや冷蔵庫内の在庫のデータを調理機材で共有すると言ったことも可能になる。

ユーザが外出していても携帯電話などから家庭内の様子を確認し、機器を遠隔操作することも視野に入れられている。帰宅直前になったら室温に応じてエアコンを操作する、風呂の準備をするなどがこれに当たる。

前述の通り、現在このような家電には http サーバが内蔵されており、PC や PDA の Web ブラウザ上からアクセスすることによりオペレーションを行う物が大半を占める。これは OS 等に非依存で特別なクライアントを必要としない点で優れているが、オーバーヘッドが大きい。

またネット家電同士の通信のためのプロトコルが現在検討されているが、これはこの特定用途向けにデザインされた軽量なプロトコルであり、デバイス・ハブと目的を異にする。

3-2 NAS

ここに来て家庭用の NAS(Network Attached Storage)が各社から発売されてきた。

これらはハードディスクに SMB や AppleTalk と言ったプロトコルを解釈するネットワークターミナルを追加した物である。Windows や MacOS などの SMB や AppleTalk が実装された OS からネットワークドライブとしてアクセスする。

I/O data 社は数年前より自社のストレージのインタフェースを i-connect という独自規格に統一してきた。ユーザは i-connect と各種ポート/バスとのコンバータを介してそれらにアクセスすることができる。

これによってストレージデバイスをポート/バスを切り替えて使用することが出来る。i-connect と LAN とのコンバータが発表された。コンバータには Linux が使用されており、その上には SAMBA や NetTalk が実装されている。OS にそれらの実装が必要であること、またネットワークストレージとして認識される点で、デバイス・ハブと異なる。



図 3-1 I/O data 社製 i-connect - LAN コンバータ LAN-iCN

3-3 iSCSI

iSCSI はコンピュータとストレージ間のデータ通信を、IP パケットで包んだ SCSI コマンドを使って IP ネットワーク上で行うためのプロトコル規格である。iSCSI では、ストレージ機器で一般的な SCSI コマンドを採用したことにより、既存のアプリケーションとの高い互換性を維持したままネットワークストレージへの対応が行えることを特徴としている。

これまでの SCSI では各種制御やデータ転送のためのコマンドとともに物理的な伝送路も規定されていたが、iSCSI では SCSI のコマンド部分を取り出し、伝送路に IP ネットワークを採用することで汎用性を高めている。iSCSI を採用することで、現在 SAN の伝送路として一般的に利用されているファイバ・チャネルの代わりとして、安価なイーサネット用のスイッチング・ハブやルータなどを使ってサーバとストレージの接続が行えるようになる。また、既存のネットワーク技術と管理手法が利用できるため、導入・管理コストが安価になることもメリットとして挙げられる。さらに伝送路に IP ネットワークを用いるため、インターネットを使って遠隔地にあるストレージにデータをバックアップするディザスタ・リカバリといった用途にも応用可能である。

iSCSI では、このように既存の SCSI 技術と IP/イーサネット技術のメリットを活かした規格化が行われており、ストレージ・プロトコルの新たな標準規格としてサーバ/ストレージ業界の期待が集まっている。なお iSCSI は、現在 IETF (Internet Engineering Task Force) [5] でドラフト規格が公開されており、標準化作業が進められている。ドラフト規格に準拠した製品も、IBM や Adaptec などからすでに発表されている。

iSCSI は SCSI バスの距離的延長を主な目的としており、かなり限定的なプロトコルである。デバイス・ハブのように統一した環境を提示する物でない。

3-4 Screen

Screen は遠隔ホストの複数のプロセスで共有するためのソフトウェアである。一つのホストの中で複数個の仮想画面を利用できる。telnet や ssh 等による接続の場合は接続が切れた時点でその中で動いていたプロセスは死ぬが、Screen その仮想画面と其中的プロセスを生かしたまま接続を切断(detach)し、その後再接続(attach)してプロセスを引き継ぐことができる。不意に接続が切れた場合にも自動的に detach するので、後から attach することで切れる直前の状態を復元することができる。

これらの機能は screen が端末エミュレータとログインシェルの間に介在し、ログインシェルを終了させることなく端末エミュレータと切り離すことで実現される。detach 場合、プロセスを screen 引き継ぎ、attach された際に新しい端末エミュレータに渡される。

Screen のように一旦接続が切断された後に再接続した場合に切断直前の環境を復元する機構はデバイス・ハブにとっても必要である。

4 デバイス・ハブ環境の設計

デバイス・ハブ環境の基本設計を行い、必要な技術を整理する。

デバイス・ハブ環境は大きくマッチングとホスト – デバイス間通信との2つに分けられる。

4-1 ネットワーク上のモデル図

ネットワークのモデルは以下の2つを想定する。

これらについては後に 5-3 で詳細を述べる。

4-1-1 Hybrid P2P モデル

Hybrid p2p と同様に各ホスト/デバイスがマッチングサーバに自身を登録し、マッチング先が見つかった時点でホストとデバイスが接続される。マッチングポリシーをサーバ側に記述することで効率的なマッチングと柔軟なポリシーの実装を可能とする。



図 4-1 ハイブリッド P2P モデル図

4-1-2 ZeroConf モデル

ZeroConf を活用することでアドホックなネットワークを形成し、ホストが求めるサービスを提供するデバイスとマッチングを図ることが可能である。

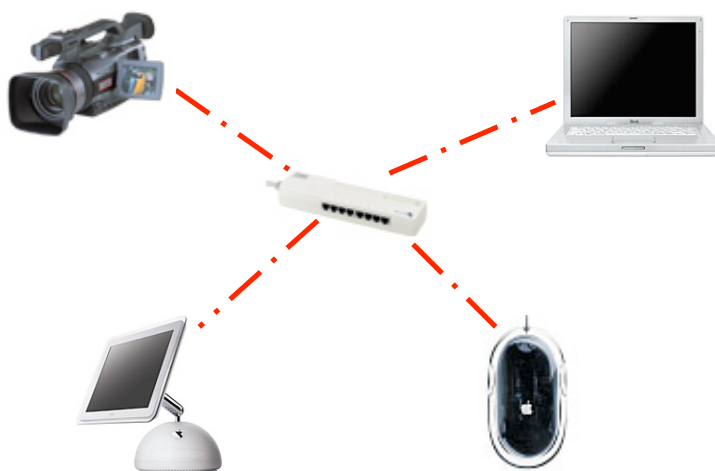


図 4-2 ZeroConf モデル図

4-3 マッチング部の設計

4-3-1 Hybrid-P2P モデル

napster などの HybridP2P のようにホスト/デバイスを結びつけるためのマッチングサーバをたてる。ホスト側/クライアント側インタフェースはネットワークが enable になった時点でマッチングサーバへ自身の情報を登録する。

デバイス側インタフェースが登録された場合はサーバがそのパラメータを参考に結びつけるべきホスト側インタフェースを探し、見つかった場合はその情報をデバイス側インタフェースに通知する。

マッチング先が見つかったデバイス側インタフェースはその後サーバを介さず、直接ホスト側インタフェースと通信する。マッチング先が見つからなかった場合はそのままプールされ、マッチングされるべきホストが見つかった時点でそのホストの情報が通知される。

ホスト/デバイスのマッチングポリシーはサーバ側にプログラムする。サーバはそのデバイス・ハブ環境の全体を把握できるため、柔軟なマッチングポリシーの実装が可能となる。

4-3-2 ZeroConf モデル

multicast DNS(以下、mDNS)は IETF ZeroConf WG と DNS extensions WG の共同作業によって生まれた小さなネットワーク上でホストヘッドホックな Naming を提供するサービスである。ネットワーク上で enable になったノードは mDNS の通知機能を使って自分の提供するサービスの種類、サービス名、IP アドレスや port、その他の必要な情報をネットワーク内のノードに知らせる。サービスの通知を受け取ったノードはそれらの情報を自信の持つ簡易 DNS サーバへ保管する。

既存の DNS からのプロトコル上の変更点を最小限にとどめ、DNS の経験のあるプログラマにとって修得しやすく、実装もしやすくなっており、またそのパケットを既存のネットワークパケットアナライザ等が特別なアップデートをすることなく理解できる

DNS Service Discovery(以下、DNS-SD)は AppleTalk の様にクライアントが DNS ベースでサービスインスタンスを探すためのサービスである。mDNS によって簡易 DNS サービスのレコードが保管された後にその DNS サーバへ問い合わせをかけることで利用できるネットワークサービスを知ることができる。

DNS-SD は既存の DNS メッセージの構造やオペレーションコード、レスポンスコード、リソースレコードタイプなどを変更せず、また新たなプロトコルを定義するものではない。以下の3つをその目的とする

- (1) ドメイン内でのサービスの問い合わせをし、また該当するサービスインスタンスのリストを受け取ることを可能とする
- (2) 目的とするサービスインスタンスを発見した後、そのサービスを実際に利用するために必要な情報(IP アドレスや port ナンバーなど)をクライアントへ提供する。
- (3) サービスインスタンスの名前を出来る限り永続化させる。IP アドレスや port ナンバーが変更になっていたとしても同一の name ベースで探したサービスインスタンスは常に同一のインスタンスの参照を可能とする

これらの目的のため、DNS-SD は出来る限りシンプルなプロトコルとし、IP ベースで動作する小さな機器にも実装できるかたちを目指している。

これら mDNS と DNS-SD、そして後述の AUTO-IP/ND を組み合わせて用いることで、ともすれば複雑になりがちな装置をネットワークに合わせて設定するプロセスを簡易にすることが出来る。中心となる DNS サーバやネットワーク管理者が不在で統一的な管理が行えない LAN やその場その場で構成される、アドホックネットワークにおいて、mDNS、DNS-SD を利用してネームサービスを実行する。

自動設定されてネットワークに組み込まれたコンピュータは、同じネットワーク内の他の装置が提供するサービスを見つけ、同時に自分が提供するサービスを他の装置に伝える必要がある。

ネットワーク内でサービスを共有するには、各装置は自分が提供する個々のサービスに固有の名前を付け、その存在をネットワーク内の他の装置に知らせる必要がある。

もちろん、通知を送った装置は、同じネットワーク内で他の装置が提供しているサービスも検索する。ユーザが装置をネットワークに接続して特定のサービスに関する情報をリクエストすると、装置はその種のサービスが提供されていないか、ネットワークに問い合わせをかける。

例えば、利用できるプリンタについてリクエストすれば装置がそれを問い合わせ、その一覧を作成する。ネットワーク内にプリントサービスを提供する装置が存在する場合には、その装置が問い合わせに対して mDNS-SD による通知を返す。問い合わせた装置が受け取った情報を自分の簡易 DNS サーバに保管し、印刷機能を持つアプリケーションからプリンタの一覧が利用できるようになる仕組みである。

デバイス・ハブ環境におけるホストとデバイスのマッチングに mDNS・DNS-SD を活用することで、中央集中的なサーバが不要でアドホックに環境を構築することができる。

4-3-3 クライアント

クライアントのネットワークコンフィギュレーションは static-configuration や DHCP を用いた statefull-configuration、また特別な状態管理サーバを用いない stateless-configuration などの方法でも良いが、ここでは前述 ZeroConf の特徴として挙げられる stateless-configuration について説明する。

これはサーバによってエンドホストごとのネットワークパラメータの状態を管理する必要がなく、またネットワークインタフェースの設定の手間無くアドホックにネットワークを形成することができるのが特徴である。

4-4-1 IPv4 の場合

IPv4 の stateless autoconfiguration は現在活発に議論が行われており、internet-draft が提出されているがまだ RFC の形で標準化されていない。

そのアドレッシングには AUTO-IP が用いられる。AUTO-IP はデバイスが予約されたプライベート アドレス セットからインテリジェントに IP アドレスを選択する方法を定義し、それにより管理ネットワークと管理されていないネットワークの間を簡単に移動可能となる。まず利用可能な DHCP サーバを確認し、見つからない場合、ARP によって割り当てられている範囲内で用いようとするアドレスが有効かチェックする。もしそのアドレスが既に使用されていれば他のアドレスでチェックを使用可能なアドレスが見つかるまで繰り返すという形でネットワークアドレスを生成する。AUTO-IP アドレスを取得後も定期的に DHCP サーバが利用可能かチェックする。

4-4-2 IPv6 の場合

IPv6 の stateless autoconfiguration は RFC2461/2462 で規定されている。

NeighborDiscoveryProtocol を利用してルータからエンドホストがネットワークに接続に必要なパラメータが通知提供される。

エンドホストはネットワークに接続する際に RouterSolicitation message をリンクローカル全ルータマルチキャストアドレスに転送し、ルータは message を受け取ると

RouterAdvertisement message をエンドホストに送信する。この message によりホストが使用すべきネットワークプレフィックス、MTU 等が通知される。

IP アドレスの下位 64bit のインタフェース ID 部はホスト自身が EUI-64 による MAC アドレスからの自動生成などで自動的に生成する。MAC アドレスからインタフェース ID を生成する際にはインタフェース ID リンク上での一意性が確保される。また、インタフェース ID の一意性を確認するために Duplicated Address Detection 機能を用いてリンク上に同一のアドレスがないかを確認する。同一のリンク上に複数のルータが存在するときはホストは複数の RouterAdvertisement message を受信する。この場合、ホストはリンクに接続された各インタフェースに複数の異なるネットワークプレフィックスをもったアドレスを割り当てる場合がある。また、RouterAdvertisement により取得した設定パラメータは恒久的なものではなく、有効な期間が定義されている。

4-4 ホストーデバイス間通信

ホストーデバイスがマッチングされた後はホストーデバイス間での直接の通信となる。

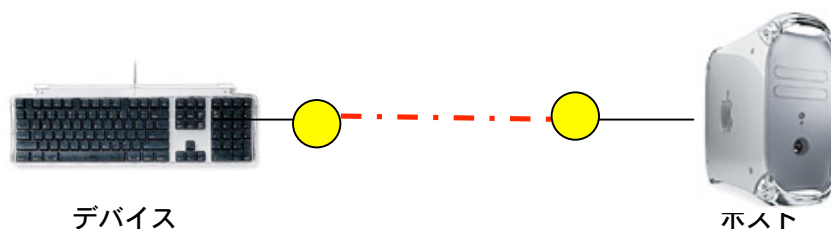


図 4-3 デバイスーホスト間通信

デバイスーホスト間のプロトコルについてはデバイス・ハブ環境では定義をしない。

ターゲットとなるポート/バスが多岐にわたり、そのそれぞれに特徴があり、それに適したプロトコルが存在するので、それをネットワーク化する際にそれぞれに対して柔軟な設計が必要になる。

デバイス・ハブ環境で統一的なプロトコルを定義することはその柔軟な設定を阻害するのでナンセンスである。

5 実装

デバイス・ハブ環境を構築し、その中で動くデバイス・ハブ デバイスとして USB キーボードを実装した。

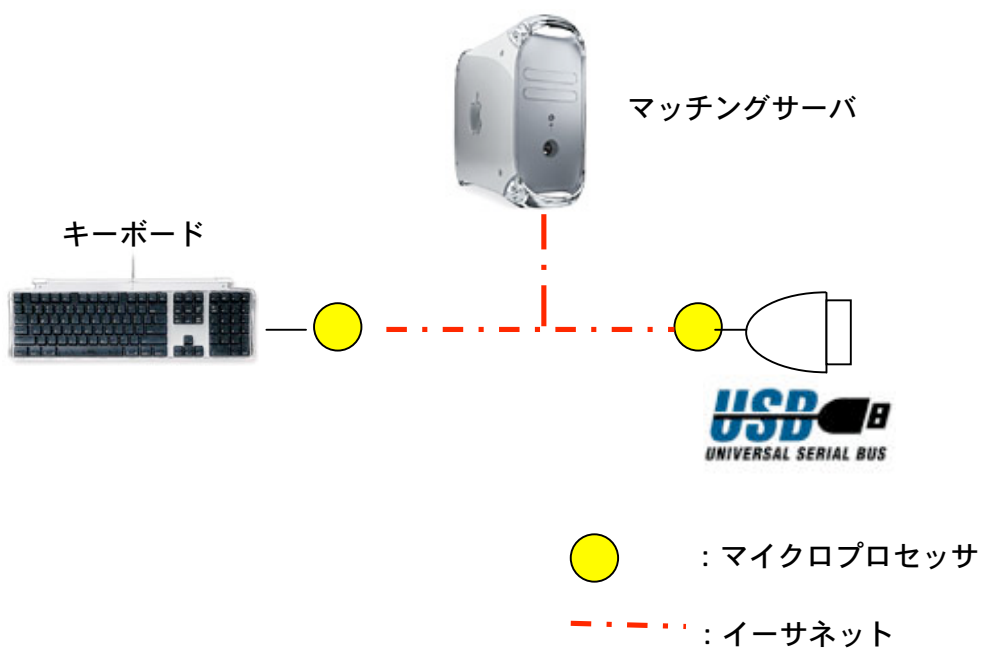


図 5-1 実装システム概要図

5-1 USB のネットワーク化について

ここで、USB のプロトコル[19][22]について簡単に解説する。

5-1-1 特徴

USB は、以下に示すような特徴を持っている

- ・ 1.5Mbps/12Mbps/480Mbps の 3 種類のバス速度
- ・ ハブを用いた自由なトポロジのレイアウト
- ・ プラグ&プレイに対応
- ・ パワーマネジメントが可能
- ・ リアルタイム系のマルチメディアデータをサポート

5-1-2 USB と Firewire

USB とよく比較される規格として Firewire(IEEE1394)がある。USB2.0 になり High Speed データ転送が可能になったことにより、Firewire に負けぬデータ転送が可能となった。しかしながら、以下の理由で USB と Firewire はアプリケーションをすみわけることになる。

USB は、あくまでも比較的低価格な PC 周辺機器の実現を狙っており、デバイス側のコストを抑えることが出来る。そして USB は PC を中心としたトポロジ(ホスト-デバイス)をとっており、テレビ、ビデオといった AV 機器や家電同士のような対等な関係で相互の通信を必要とする機器には対応できない。

一方 Firewire は開発当初より高速化を目指しており、現在でもギガビット 1394 と呼ばれる 800Mbps や 1.6Gbps,3.2Gbps が規格化されようとしている。ハイエンドもしくは家庭内の AV システムといった用途に適する仕様になっている。

5-1-3 トポロジ

USB 仕様には複数の観点のバストポロジが存在する。

USB 仕様に置いて、一つの USB システムの中に複数のホストを入れたり、通信経路に複数のバスを持たせたり、ループ状のリンクは出来ない。上流と下流という形でケーブルの両端のコネクタを異なる形状にすることによりその可能性を防いでいる。

5-1-3-1 物理的なバストポロジ

ホストを唯一のバスマスタとしてバス分岐点にハブを配置して末端に USB デバイスがぶら下がるという階段状のスター型バスを構成する。1つの USB システムの中では1台のホストと合計最大 127 台のノード(ハブ/デバイス)を接続できるハブは5台までチェーンでき、ハブ-ハブ間、及びハブ-デバイス間をケーブルで接続する。

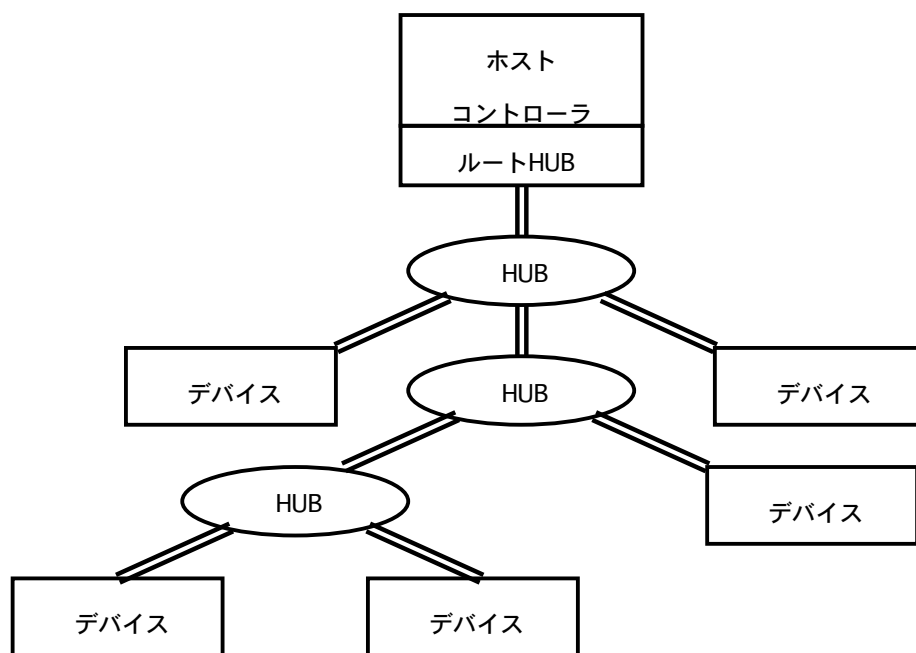


図 5-2 USB 物理的トポロジ

5-1-3-2 論理的なバストポロジ

前述の通り、バストポロジは階段状のスター型をとる。論理的なバストポロジにおいてはハブの存在が削除され、1つのホストと複数のデバイスを接続している1対多通信のように抽象化される。この抽象化は、ホストとハブの処理によって実現されている。よって USB デバイスを利用するホストのデバイスドライバを作成する場合にはデバイスごとに独立したソフトウェアとして用意する。

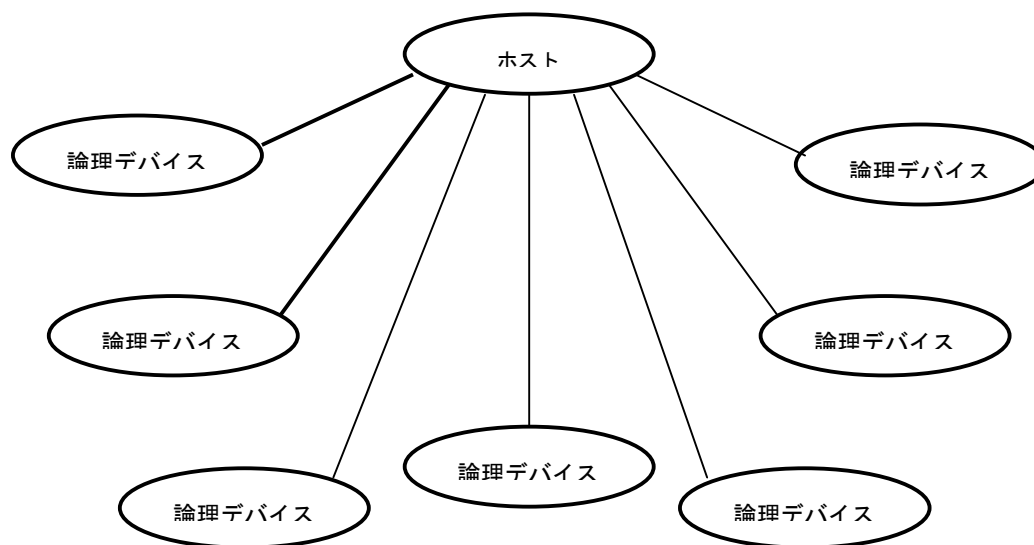


図 5-3 USB 論理的トポロジ

5-1-4 データ転送タイプ

USB システムの物理的なバストポロジから、論理的なバストポロジを実現するために、最下層のプロトコルでは、ホストと各デバイスが時分割で通信して、ホストがスケジューリングを行う。このスケジューリングはホストのシステムに任されているが多様なデバイスが混在するシステムをとる USB デバイスにおいてはバス利用率とそれぞれのデバイスのパフォーマンスを下げないようにアプリケーションの特徴に合わせて選択できる 4 種類のデータ転送タイプを定義している。

5-1-4-1 コントロール転送

勃発的で非周期的な通信のうち、リクエストレスポンス形態の双方向通信である。標準デバイスリクエストと呼ぶコントロール転送を使用したやりとりが定義されている。これは主にホストがデバイスをプラグ&プレイするために使用され、全ての USB デバイスは、このリクエストに対応する必要がある。他の転送タイプに比べて、コントロール転送のプロトコル上の通信オーバーヘッドが大きく、ハイスピードやフルスピードでの大きなデータの転送には向かない。けれども、他の転送タイプでの通信中にデバイス制御が可能である。

5-1-4-2 バルク転送

勃発的で非周期的な通信のうち、遅延が問題にならない、大量のデータを転送する用途で用いる。例えばプリンタの印字データやスキャナのイメージデータ、ストレージ機器のデータなどで用いる。他の転送タイプよりスケジューリングの優先度が低くなるが他の転送の空き時間を効率的に用いることが出来る。

5-1-4-3 インタラプト転送

非同期で低頻度のイベントを、ホスト-デバイス間で通知するために使用する。たとえば、マウスやキーボードなどの入力データがある。この転送タイプの名前から連想されるイメージとは異なり、インタラプト転送は周期的なホストからのポーリングによって処理され、そのポーリングの間隔はデバイスからホストに申告する。

5-1-4-4 アイソクロナス転送

連続的で周期的な通信に使用する。通信経路を確立した後は限定的なレイテンシーで一定の転送レートが保証される。動画や音声データのようなリアルタイム性を必要とするストリーミングデータで使われる。この転送タイプはエラーを検出するための CRC エラーチェックのみが行われ、再転送手順を持たない。ただし USB システムは、信頼性が高いバスでほとんどの転送が成功し、現実のシステムではあまり問題にならないようになっている転送データを保証しないことにより再送のための遅延が発生せず、転送レートを保証することが出来る。

5-1-5 バス速度

現在 USB では 480Mbps のハイスピード(HS)/12Mbps のフルスピード(FS)/1.5Mbps のロースピード(LS)の3種類のバス速度が定められている。

ロースピードデバイスを非常に低コストで実現できるように最大限の配慮がなされている。このため、ロースピードデバイスはコントロール転送とインタラプト転送のみが可能である。

表 5-1 転送モードごとに使用可能な転送タイプ

バス速度	コントロール 転送	バルク転送	インタラプト 転送	アイソクロナ ス転送
ロースピード (LS/1,5Mbps)	○	×	○	×
フルスピード (FS/12Mbps)	○	○	○	○
ハイスピード (Hs/480Mbps)	○	○	○	○

5-1-6 エンドポイント

デバイスとホストの間での通信経路の最小単位をエンドポイントという。

USB の通信は、ホストとエンドポイントの間で行われる。ホストは一つのシステムの中でユ

ニークなアドレスをデバイスに割り当て、デバイスのエンドポイントと通信する。なお、一つのデバイスに複数のエンドポイントを持たせることが出来るので複数の転送を組み合わせで一つのデバイスの機能を実現できる。ホストが発行するトークンパケットの ADDR、ENDP フィールドはアドレスとエンド本とを示す。

7bit 分の ADDR フィールドがあり、ホストは 1~127 のアドレスをデバイスに設定する。アドレス 0 はデフォルトアドレスと呼ばれ、初期状態のデバイスがホストからのリクエストを受け付けるための物として扱われる。

デバイスは 0~15 の番号を持つエンドポイントを持つことが出来るが、必ずコントロール転送用のエンドポイント 0 を持つ必要がある。標準デバイスリクエストはエンドポイント 0 で行われる。

5-1-7 インタフェース

デバイスは複数のエンドポイントを持つことが出来るがそれらをインタフェースという論理的な単位で機能的にグルーピングする。一つのデバイスが複数のインタフェースを持つことも出来るし、それらの集合がデバイスの構成になる。さらに、インタフェースの代替設定や複数の構成を定義することにより複雑なデバイス構成情報を作成することが出来る。

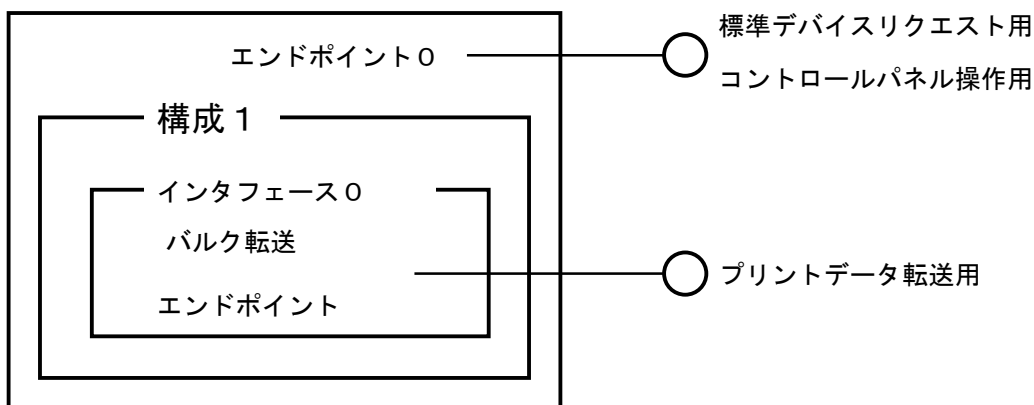


図 5-4 単インタフェースを持つ USB プリンタデバイス構成

インタフェースの代替設定は、インタフェースの持つエンドポイントの構成や特性を切り替えるために使用する。例えばデバイスやホストなどの状態によりエンドポイントのペイロードを動的に変化させたい場合、代替設定の異なる複数のインタフェースを用意することで対応出来る。以下の図は単方向/双方向プリンタの対応可能なデバイス構成の例である。

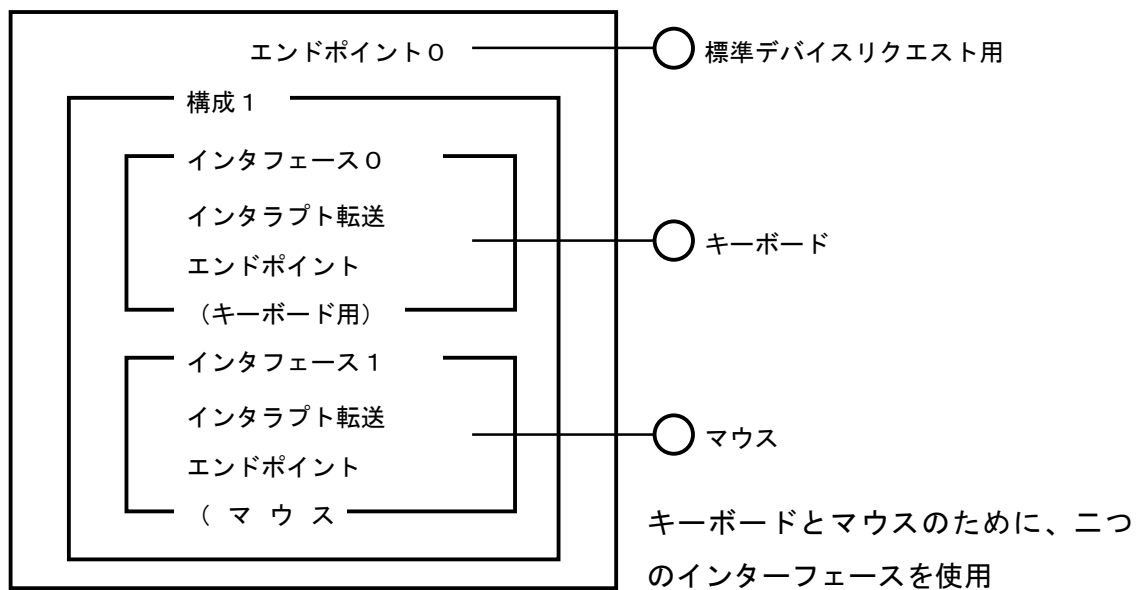


図 5-5 複数のインターフェースを持つ USB プリンタデバイス構成

このように複数の構成を持たせることによりエンドポイントやインターフェースの組み合わせが異なるようなデバイス構成を排他的に切り替えるようなデバイスが実現できる。

5-1-8 プロトコルデザイン

USB システムにおいては、通信はホストの USB のプロトコルスタックが主導権を握っている。即ち、たとえデバイス側からデータを転送する場合においてもホストがデバイスに対してバスの使用权を与えてからデバイスがデータを送信します。

USB バス上では、パケットという単位でデータを送受信する。いくつかのパケットによってトランザクションという通信の単位を構成する。さらに、そのトランザクションが集まって、一つの転送を構成する。

ホストは複数の転送を同時に処理することにより、複数のデバイスをサポートしたり一つのデバイスで複数の転送を組み合わせたシステムを構築することが出来る。

5-1-9 パケット

ホストやデバイスがバスの時間を確保するデータの単位。同期(SYNC)パターンで始まり、EOP(End Of Packet)までの時間的に連続したビット列で構成される片方向の通信単位である。各パケットの頭には 8bit の SYNC パターンと 8bit の PID(パケット識別子)が含まれる。

5-1-10 トランザクション

ホストがスケジューリングする単位で、コントロール、バルク、インタラプト、アイソクロナス転送をする単位になる。

トランザクションの種類には以下のような物がある

- ・ IN トランザクション
- ・ OUT トランザクション
- ・ セットアップトランザクション
- ・ アイソクロナストランザクション(IN/OUT)

5-1-11 クラス

USB デバイスは、それぞれの使われ方やデータ転送形式などによっていくつかのクラスに分けられている。代表的なクラスを以下に挙げる。

- ・ ヒューマンインタフェースデバイスクラス

キーボードやマウス・タブレットなどがこれに属する。

- ・ オーディオクラス

サウンドプロセッサやスピーカーなど、オーディオデバイスがこれに属する。

- ・ イメージングクラス

スキャナ等がこれに属する

- ・ マスストレージクラス

ハードディスク等がこれに属する

- ・ プリンタクラス

プリンタがこれに属する

- ・ コミュニケーションデバイスクラス

モデムやネットワークインタフェースデバイスなどがこれに属する

- ・スマートカードクラス

メモリカードリーダー名護がこれに属する

- ・フィジカルクラス

フォース・フィードバック・ジョイスティックなどがこれに属する。

5-2 USB キーボードの実装

5-2-1 USB プロトコルについて

今回、USB プロトコルを Ethernet 上に展開するために以下の図のような機能を実装した。

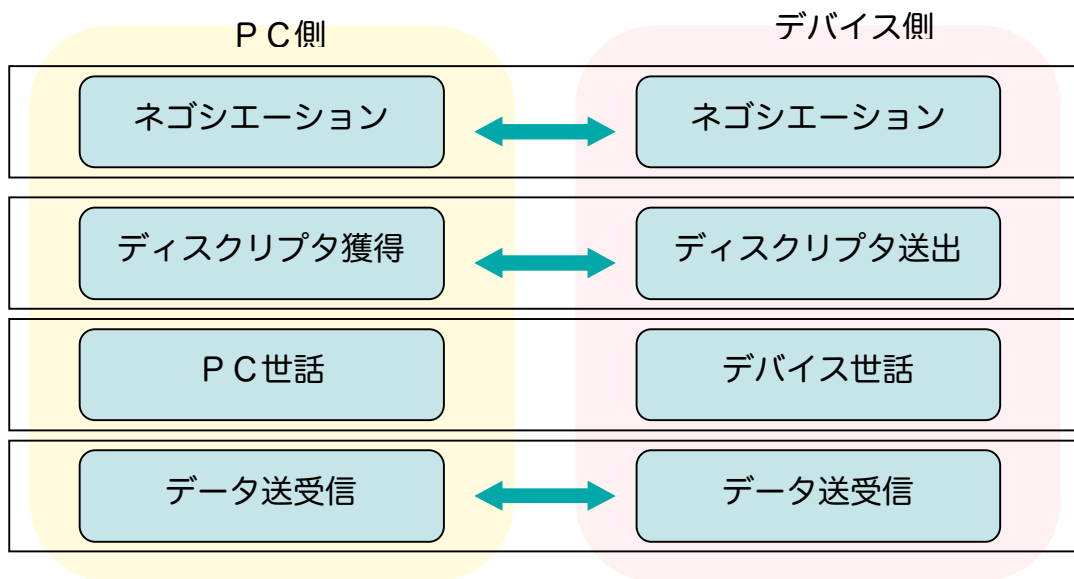


図 5-6 USB スタック図

通信は大まかに図の上から下に向かって進む。

(1)マッチングが行われた後、まずホスト側とデバイス側とでのネゴシエーションを行う。
ネゴシエーションはデバイス側からホスト側へ connect する。その際に PC が accept し、仮想経路が確立される。

(2)ディスクリプタ交換

その後、USB スタックに乗っ取りディスクリプタを交換する。デフォルトパイプを通じて PC からデバイスへディスクリプタのリクエストを行い、それにデバイスが答えるという形でコンフィギュレーションが行われる。個々で交換されたディスクリプタを元にしてホストはデバイスの使用条件を設定する。デバイス側のインタフェースはあらかじめデバイスから各種ディスクリプタを獲得しておき、ホスト側からのリクエスト時にはそのコピーを送出する。コンフィギュレーションが終了するとホストと各エンドポイントの間に必要な数のパイプが設定される。

デフォルトパイプはコントロール転送を行うためにこの時点で破棄されない。

(3)アプリケーションにおける通信

以後、各インタフェースはホスト/デバイスとの通信と他方のインタフェースとのコミュニケーションを繰り返す。ホスト側インタフェースはホストからのリクエストに応じてデバイス側インタフェースにリクエストを転送し、デバイス側インタフェースはリクエストを受け取ったらデバイスとコミュニケーションし、適切なレスポンスをホスト側インタフェースへ返す。

5-3 インタフェースノード実装について

インタフェースノードには OS として mozuku を用いた。

5-3-1 mozuku について

mozuku は μ -iTron4.0 に準拠した OS である。

H8 マイコン、Linux、netBSD 上での実装が存在し、 μ -iTron4.0 にて定義されている標準 API を全て備える。

ipv6 スタックを装備し、Neighbor Discovery、mobile-IP(based on draft19)などといった IPv6 に要求される機能のほとんどとそのための API を備える。ただし、現在のところ IPsec は未実装。

5-3-2 TRON について

TRON [10]とは The Realtime Operating system Nuclear の略で、1984 年に東京大学の坂村健によって開始された産学協同のプロジェクトである。

TRON プロジェクトの究極の目的としては HFDS (Highly Functionary Distributed System) 超機能分散システムの実現にある。HFDS の構成要素として BTRON, CTRON 等のサブプロジェクトがある。

TRON の特徴としてオープンアーキテクチャという点が挙げられる。標準としてのしぼりは弱く、定義されているのは OS のインタフェース部分だけで、実装は何でも良い。UNIX でいうところの POSIX や、CORBA に似ている。TRON 協会は仕様の策定と仕様適合性の確認を行い、実装は行っていない。

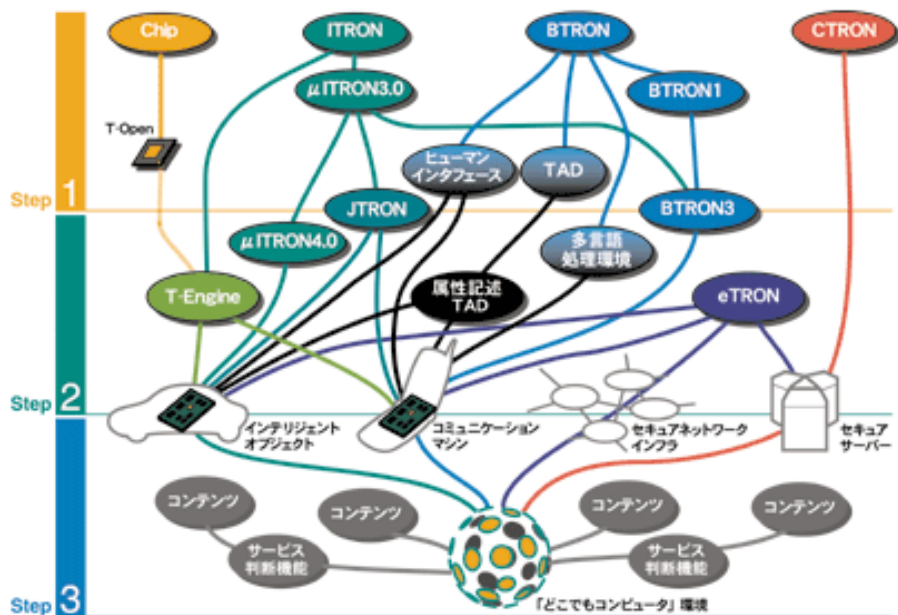


図 5-7 TRON HDFS

サブプロジェクトとしては次のようなものがある。

基礎プロジェクト

ITRON
JTRON (ITRON & JAVA)
CTRON
TRON Chip
MTRON
ETRON (IC カード)

応用

TRON 電腦住宅
トロン電腦ビル
デジタルミュージアム

この分野のトレンドとしては

- ・ OS チップ化
- ・ Embedded TCP/IP の実装

- ・ TRON GUI の定義

といったものがある

また、その設計は以下の点を重視して行われている

- ・ ハードウェアの過度の仮想化を避け、ハードウェアに対する適応化を考慮する

ハードウェアの持つ性能を最大限に発揮させ、高いリアルタイム性能を得るためには、仕様作成にあたって、ハードウェアを過度に仮想化することは避けなければならない。ハードウェアに対する適応化とは、ハードウェアの持つ性能や性質に応じてリアルタイム OS の仕様や内部の実装方法を変え、システム全体としての性能向上をはかることをいう。

具体的には、ITRON 仕様においてハードウェアによらず標準化すべき事項と、ハードウェアの持つ性能や性質に応じて最適になるように決定してよい事項を明確に分離した。標準化した項目には、タスクのスケジューリング規則、システムコールの名称と機能、パラメータの名前・順序・意味、エラーコードの名前と意味などが含まれる。一方、標準化すると実行時性能の低下につながるような部分については無理に標準化せず、標準化・仮想化による性能の低下を避けるように配慮した。具体的には、パラメータのビット幅や割り込みハンドラの起動方法については、実装毎に決める方針としている。

- ・ アプリケーションに対する適応化を考慮する

アプリケーションに対する適応化とは、アプリケーションに必要となるカーネルの機能や要求される性能に応じて、カーネルの仕様や内部の実装方法を 変更し、システム全体として性能向上をはかるアプローチをいう。組み込みシステムの場合、OS のオブジェクトコードはアプリケーション毎に生成するため、アプリケーションに対する適応化が特に有効に働く。

具体的には、カーネルが提供する各種の機能をできる限り独立させ、アプリケーション毎に必要な機能だけを用いることができるように考慮して、仕様 の設計をおこなっている。実際、多くの μ ITRON 仕様カーネルは、カーネル自身がライブラリの形で提供されており、アプリケーションプログラムとリンクするだけでカーネルの必要なモジュールだけが組み込まれる仕組みになっている。また、システムコールの単機能化により、必要な機能のみを組み込むことを容易にしている。

・ソフトウェア技術者の教育を重視する

組み込みシステムにおいては、あるシステム用に開発されたソフトウェアがそのまま次に開発するシステムに使えることはまれであり、ソフトウェアの 互換性やポータビリティは、それほど重視されない傾向にある。それよりも、カーネル仕様の標準化によって、ソフトウェア技術者の教育が容易になることや用語や概念の統一を通じて技術者間の意志疎通がスムーズになることが重要と考えられる。

ITRON 仕様では、ソフトウェア技術者の教育を重視し、標準化を通じて一度覚えた事が広く活用できるよう配慮している。具体的には、仕様における用語の使い方や、システムコール等の名称の決め方などは、できる限り一貫性を持つように配慮した。また、教育用テキストの作成にも力を入れている。

・仕様のシリーズ化やレベル分けをおこなう

多種多様なハードウェアへの適用を可能にするため、仕様のシリーズ化やレベル分けをおこなう。これまでに作成したリアルタイムカーネル仕様の内、 μ ITRON 仕様 (Ver 2.0) は主に 8 ～16bit MCU を用いた小規模なシステム向けに作成したもので、ITRON2 仕様は 32bit プロセッサ向けの仕様である。またこれらの仕様の中でも、機能毎の必要度に応じたレベル分けをおこない、カーネルを実装する際には必要度の高い機能のみを実装すればよいものとしている (図-2)。最新の μ ITRON3.0 仕様では、システムコールのレベル分けにより、1つの仕様で小規模なプロセッサから大規模なプロセッサまでをカバーしている。

また、ネットワークで接続された分散システムのための仕様や、マルチプロセッサシステムのための仕様も、一連の ITRON 仕様シリーズの中で標準化作業をおこなっている。

・豊富な機能を提供する

カーネルが提供するプリミティブを少数に限定するのではなく、性質の異なる豊富なプリミティブを提供するというアプローチを取る。アプリケーションやハードウェアの性質に適合したプリミティブを利用することで、実行時性能やプログラムの書きやすさの向上が期待できる。

mozuku は様々な TRON シリーズの中でも μ -ITRON の実装である。ITRON とは組込 OS 向けのプロジェクトである。小型軽量で、実際の実装では OS サイズが数 kb から数十 kb 程度に収まる。トヨタ自動車 また炊飯器のファームウェアや携帯電話のコアなどに使用され、特に国内で組み込み OS のデファクトスタンダードとなっている。ITRON は組込向けということでユーザインタフェースについては定義されていなかったが最近では一般家電や携帯電話でも液晶などの表示部分を持つものが多くなっている。そのため ITRON で画面出力などのインタフェース仕様を定める方向に進んでいる。

今回の実装ではネットワーク的な距離が近い場面を想定したのでインタフェース間の通信に udp を用いた。TCP での通信は、インタフェースのプログラミングが時間制約的に困難であり、また今回のようにタイミングがクリティカルな場面での実装には向かないと考えた。今後、メトリックの高いエンドポイント同士の通信やより信頼性の要求される場面での実現のために TCP での実装についても検討してゆく。

5-4 インタフェースノード実装環境

今回の実装に当たっては以下を用いて行った
以下のいくつかについては詳細を後述する。

マイコン	hitachi 製 16bit マイクロコンピュータ H8/3069
ネットワークインタフェースチップ	RealTech 社製 RTL8019AS
USB コントローラ	ナショセミ社製 USBN9603
プログラミング言語	C 言語、アセンブラ
クロスコンパイル環境	Kondara Linux (kernel version:2.4.18) binutils-2.13 GCC for H8300-hms 3.1-1 newlib-1.10.0 gdb-5.2.1

5-4-1 H8 について

H8 マイコン[13]は日立製作所が開発した 8bit/16bit マイクロコンピュータである。
メモリやシリアルインタフェースなどの周辺機器を内蔵している。
様々なニーズに対応するため、いろいろな品種を用意し、そのシステムに求められる周辺機器
をシリーズごとに特徴を持たせて内蔵している。H8 マイコンであれば、コンパクトで低消費
電力なシステムを構築することが可能である。命令に上位互換があるのも大きな特徴であり、
下位シリーズのプログラムをそのまま利用することができる。パッケージも SDIP、QFP、CSP、
TQFP と各種用意されており、用途によって使い分けられる。
改めて主な特徴を以下に挙げる

- ・ 豊富なラインナップ

H8 マイコンはその利用目的に合わせた必要かつ十分な機能と性能を備え、豊富なラインナ
ップの中から内蔵メモリや周辺機能で選択できる。

- ・ CPU のスケーラビリティ

異なるシステムでも優れた環境で開発を進めることが出来る。例えばオブジェクトレベルで
命令互換性を保ちながら 8bit core から 16bit core が製品化されており、さらには 32bit core

も計画されている。

- ・高い演算能力

比較的高周波数での動作が可能であり乗除算や積和演算も高速に実行できる。

- ・低消費電力

バッテリーを小型化でき、応用機器を小型軽量化することが出来る。H8 マイコンには消費電力を大幅おさえたローパワー製品もあり、さらに動作しなくてもかまわない内蔵周辺機能は一時的にクロック供給を停止するモジュール・スタンバイ機能もある。

- ・内蔵フラッシュ ROM

応用システムを考えると、マスク ROM だけの製品では困る。H8 は内蔵 ROM にフラッシュ ROM を利用した製品もあり、ようよう製品出荷後にシステム・プログラム全体を書き換えることが可能である。プログラムの一部を RAM でエミュレーションする機能もあるのでパラメータを現地で調整しながらシステムを構築することも可能

- ・オンチップ・デバッグ機能

ワンチップマイコンの場合、開発環境にはフル規格の ICE が必要であった。しかし H8 にはオンチップでバッグ機能を搭載した製品があり、5 本程度の信号端子でエミュレータと接続できる。オンチップ・デバッグ機能を利用したエミュレータ(E10A や E10T)を接続することで、C 言語のソースコードデバッグも可能。

今回用いた H8 3069 の主要な仕様/ブロック図は以下の通り

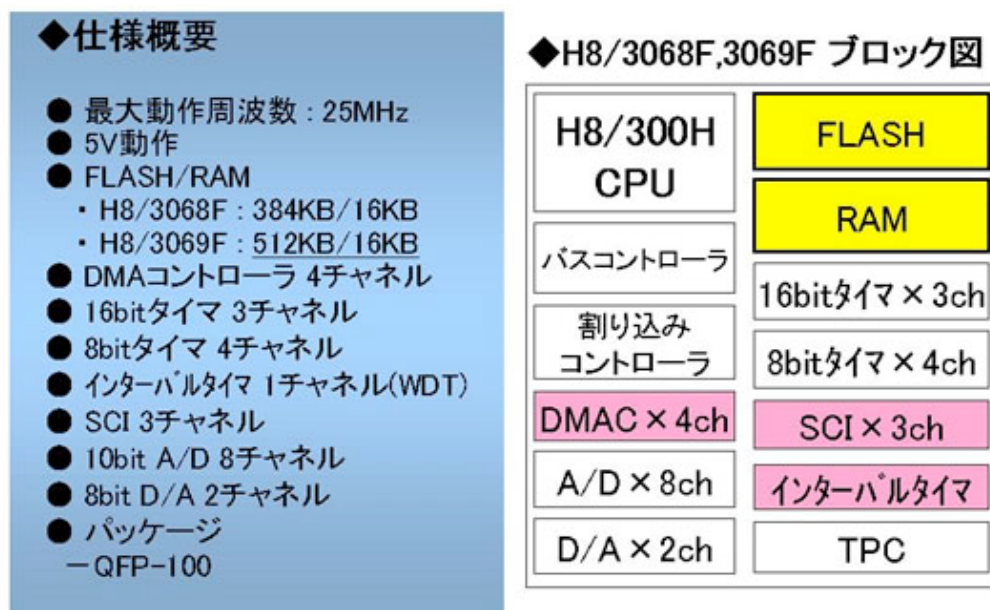


図 5-8 H8-3069 ブロック図

前身の H8 3068 より ROM が 512Kbytes に拡張され、また内蔵フラッシュROMへの書き込み時間が改善され、使いやすいチップとなっている。

今回、動作周波数は 20MHz にて実装した。

5-4-2 RTL について

Ethernet コントローラとして比較的情報ソースの多い NE2000 互換のコントローラを選出した。ただしネットワークコントローラにはメモリなどの周辺ペリフェラルが必要であり、また部品の入手も困難なために、これらの問題をクリアする NE2000 コントローラを探した。そこで周辺ペリフェラルを内蔵し、個人で入手しやすい RealTech 社製の RTL8019AS を用いた。RTL8019AS は ISA バス用の LAN コントローラ IC で Ethernet に必要な MAC/PHY といった機能の他に 16K bytes のバッファ RAM を内蔵している。

RTL8019 は H8 と相性が良く、特にインタフェース回路なしで直結できる。

5-4-3 USBN9603 について

USB コントローラとしては USBM9603[14]の他に AMD 社製 Am186CU[15]やサイプレ

ス社製 cAN2131SC[16]などが候補としてあげられる。それぞれに特色を持っており、そのどれも使いでのあるチップであるが、今回は入手可能性等を考慮して USBN9603 を採用した。

USBN9603 はフルスピード対応の USB コントローラである。28pin の SOP タイプという小降りのパッケージでありながらコントロールエンドポイントの他にユーザ用として 6 個(送信用 3 個と受信用 3 個)のエンドポイントを持つ。

エンドポイントの FIFO サイズは送受信とも 32bytes の物が 2 つずつと 64bytes の物が 1 つずつ用意されている。各エンドポイントとも USB の各転送モード(バルク・インタラプト・アイソクロナス)に対応している。割り込みは当然のこと、DMA 転送も行えるようになっている。

ホストインタフェースは、以下の 3 種類をモード設定ピンで選択できるようになっており、通常のバス出力の CPU の他、PIC 等のワンチップマイコンにも接続できるようになっている。

- ・ NON-Multiplexed mode

ISA バスのようなアドレスバスとデータバスが分離されたバスに接続することを考えたモード。このモードでは I/O アドレスを 2 つ使い、アドレスポートで内部レジスタの番号を指定してデータポートでデータのリード/ライトを行うという手順になる。

- ・ Multiplexed mode

PCI バスや 8086 の様に、アドレスバスとデータバスが共有されており、時分割で使用されるようなバスに接続することを考慮したモード。

- ・ Microwire/PLUS interface モード

シリアル伝送で各レジスタのリード/ライトを行うもの。SK(Serial Clock)に同期させて SI(Serial Input)

ピンからデータ入力・SO(Serial Out)ピンからデータ出力をする。シリアル伝送でデータの入出力を行うので速度的には不利だが PIC マイコンのようなワンチップマイコンに接続するには比較的手軽なインタフェースといえる。

5-4-4 クロスコンパイル環境

今回は Linux 上に H8 クロスコンパイル環境を構築した。

クロスコンパイルには以下の GNU 製のフリーソフトを利用した。

- ・ バイナリユーティリティ

アセンブラなどのバイナリユーティリティ

binutils-2.13 を H8 用に作って使用

(<http://sources.redhat.com/binutils/>)

- ・ C 言語コンパイラ

Linux 上で H8 上でのプログラムをコンパイルするためのコンパイラ

gcc3.1-1 を H8 用に作って利用

(<http://gcc.gnu.org/>)

- ・ C 言語標準ライブラリ

H8 用の C 言語ライブラリ。

newlib1.10.0 を H8 用に作って使用

(<http://sources.redhat.com/newlib/>)

- ・ デバッガ

作成したプログラムを Linux 上でシミュレートし、またリモートデバッグするためのツール

gdb-5.2.1 を H8 用に configure して使用

(<http://sources.redhat.com/gdb/>)

その他、make など Linux 標準的な開発環境を併用。

5-5 実装項目

5-5-1 サーバ

サーバは Hybrid-P2P モデルの実装を行った。

ホスト側が登録された時点でその IP アドレスをデータベースへ記録し、デバイス側が問い合わせをかけてきたところでマッチングされるべきホストを検索し、該当するホスト側の IP アドレスを返すことでマッチングを行う。

マッチングされたホスト/デバイスのレコードには、それマッチングされた時点で相手の IP アドレスなどが update される。サーバからホスト/デバイス両方のインタフェースに定期的にパケットを送信し、リプライが来なかった(タイムアウトした)場合、そのホスト/デバイスは無効と見なされ、テーブルから delete される。

データベーステーブルは

- | | |
|--------------|---------------------------|
| ・ホストテーブル | 登録されたホスト側インタフェースの情報を保持する |
| ・デバイステーブル | 登録されたデバイス側インタフェースの情報を保持する |
| ・デバイスタイプテーブル | デバイスの種類やクラスなどの情報を保持する |

の3つを用意した。

デバイスタイプデータベースはデバイステーブルに関連づけられている。

サーバの実装/運用を行ったハードウェア環境は以下の通り。

CPU	Intel	pentium!!! 650MHz
メモリ	512MB	
ハードディスク	20GB IDE	
NIC	corega	100base-T

サーバの実装を行ったソフトウェア環境は以下の通り。

OS	Kondara Linux (kernel version:2.4.19)+ USAGI stable release4
----	---

使用言語	Java
コンパイラ	Sun JDK J2SE 1.4.0
Java 追加 API	JDBC (mm.mysql-2.0.14)
Database	Mysql-3.23.49
待機 port	8270

但しこれらの条件は、筆者の身近にあったので用いたにすぎず、それぞれの特有な機能や性能を期待したものではない。

5-5-2 クライアント

クライアントはホスト側、デバイス側ともにネットワークスタックと USB スタックの実装の大きく2つに分けられる。USB スタックに関してはホスト側がデバイスシミュレート、デバイス側がホストシミュレートの形をとり、それぞれの実装を行った。ネットワークスタックに関しては IPv6 による stateless-autoconfiguration を用いて自身のネットワークパラメータを設定するように実装した。

クライアント同士の通信には 8271 番のプライベートポートを用いた。

また、マッチングサーバへの自身の登録とマッチングサーバからの監視への応答用に 8270 番のプライベートポートを用いた。

5-6 実装上のポイント

今回の実装ではホスト/デバイス両側のインタフェースマイコンに H8/3069 を用いた。開発中はデバッグ用に LCD ディスプレイや LED を付加し、RAM を追加した。ROM への書き込み回数は限られているためそこにはモニタプログラムを書き込み、プログラムは RAM へ転送し、そこから起動する方法をとった。これにより同じ石に何度でもプログラムを書き込み、実機テストを行うことが出来る。

USB N9603 からのデータの取得は IRQ による割り込み駆動でおこなった。USB キーボードはインタラプト転送を用いており、キーボードからのデータは断続的でまれである。よって H8 マイコンでのキーボードからのデータの獲得は割り込みによって取得する設計にした。これにより USB N9603 からの信号の監視のためのパフォーマンスの犠牲をポーリングする場合に比べ最低限におさえることが出来る。

前述の通り、デバイスのディスクリプタはあらかじめデバイス側インタフェースがデバイスから獲得しておき、コピーを持っておく。実際のホストとのコンフィギュレーションの際にはそのコピーを元に行う。

5-7 完成品

完成品の外観は以下の通り

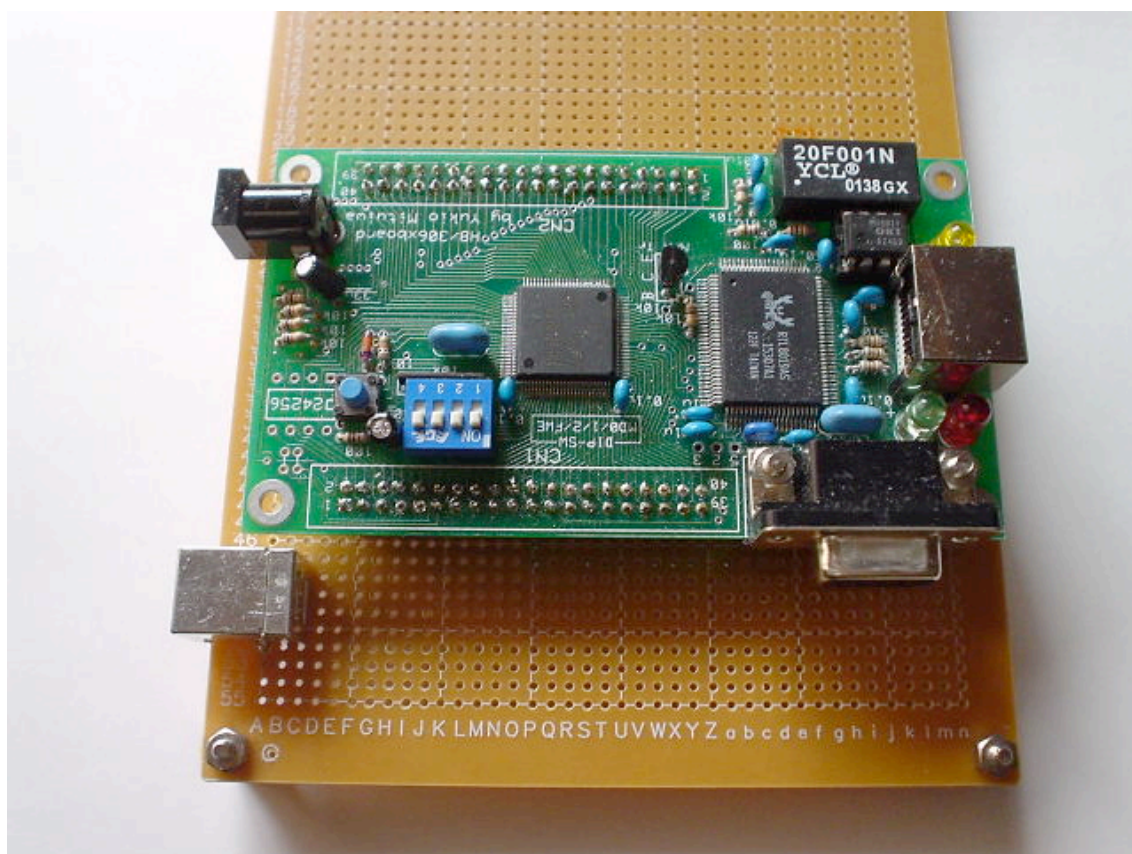


図 5-9 インタフェースノード(ホスト側)

デバイス側インタフェースノードは USB 端子の形状が異なる以外、ホスト側と同じであるので写真は割愛する。

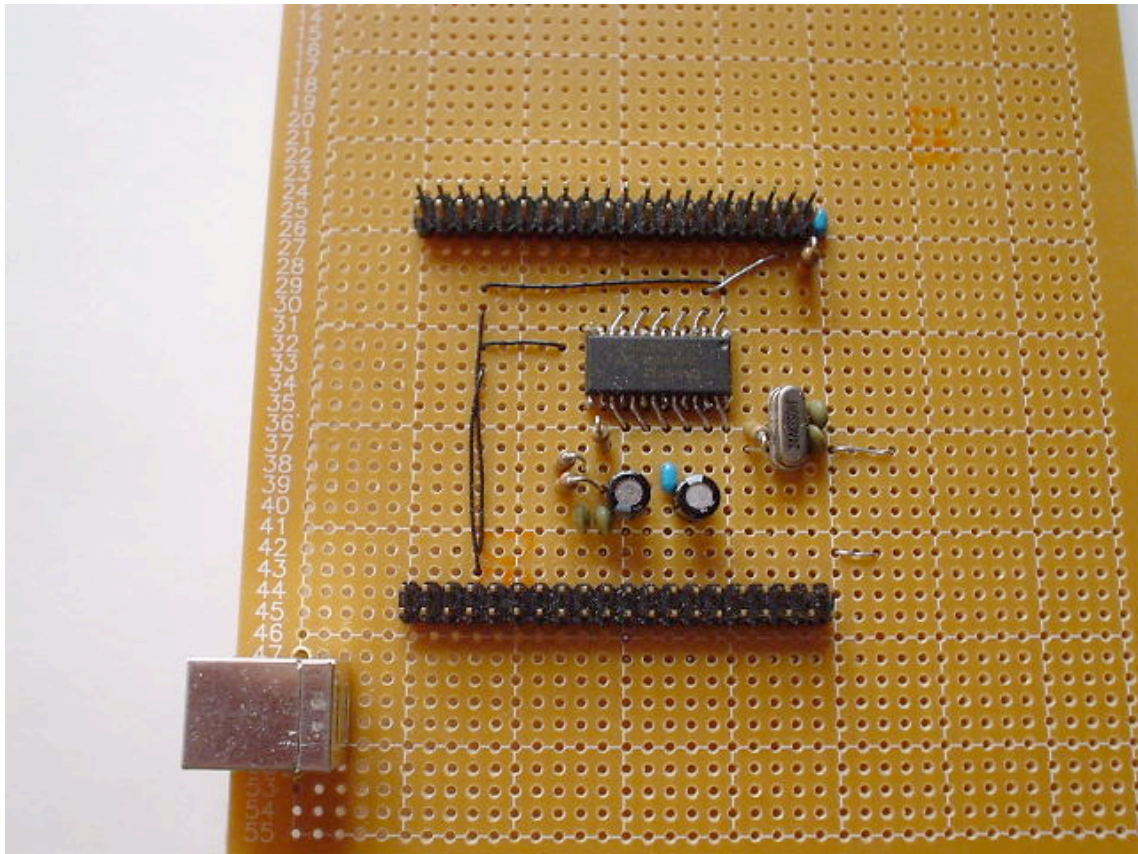


図 5-10 USB インタフェース部

6 評価

6-1 評価環境について

評価に際しての機材は以下の通り

USB インタフェースコントローラ	UHCI
ホスト側 OS	windows2000
キーボード	Microsoft 社製 natural keyboard USB モデル PFU 社製 Happy Hacking Keyboard (PD-KB200B/U)

以下の環境で動作を確認した。

まず、ホスト側へ直接(ハブを通さずに)インタフェースを接続した。

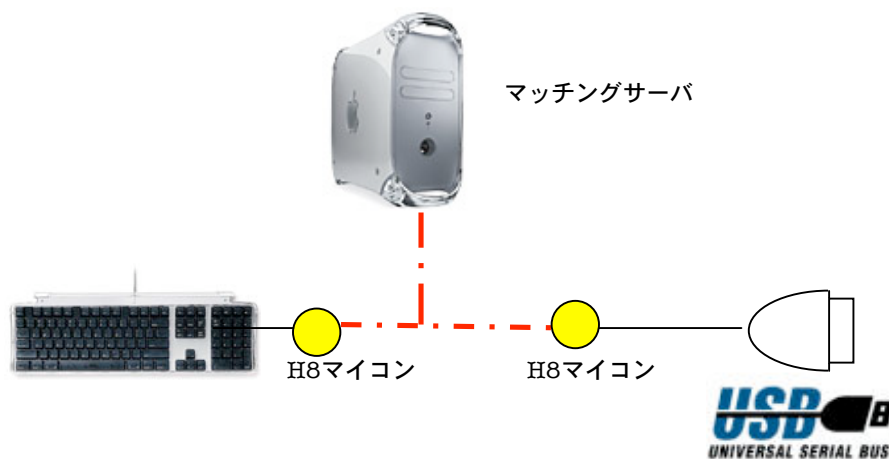


図 6-1 実験トポロジ図

両方のキーボードとも Windows の認識処理の後、通常のキーボードと同様に使用することが出来た。

数時間、通常使用したが信号を落としたことはなかった。

次に、ホスト側インタフェースと PC の間に USB ハブ(REMEX 社製 RUS-H105)を接続した。

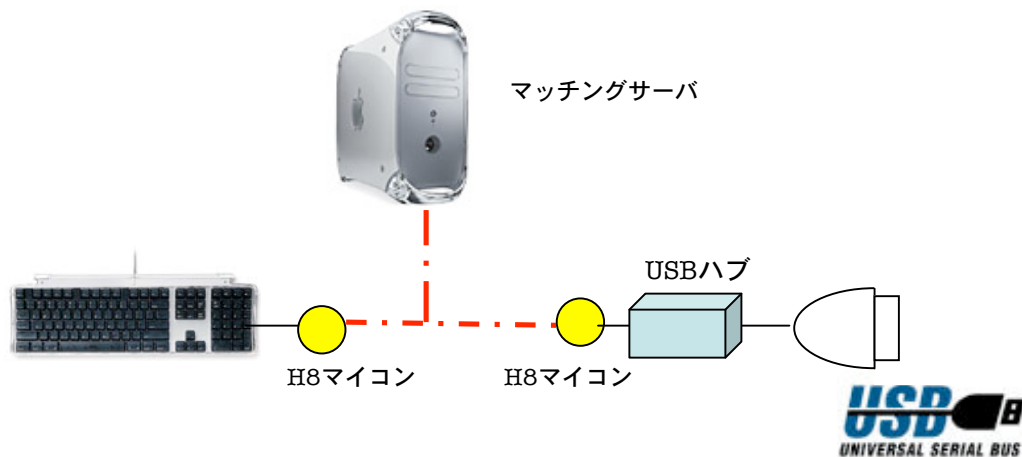


図 6-2 実験トポロジ図

両方のキーボードとも Windows の認識処理の後、通常のキーボードと同様に使用することが出来た。

数時間、通常使用したが信号を落としたことはなかった。

USBハブの別のポートへサウンドプロセッサ(ONKYO 社製 SE-U55)とマウス(MicroSoft 社製 IntelliMouse)を接続して同様の操作を行ったがどのデバイスも正常に動作した。

さらに、現実にデバイス・ハブ環境のにおいて起こりうる様々な場合を想定して実験を行った
以下の実験は USB ハブを介さないで行った。

- ・ 通常動作中、ホストから USB インタフェースを引き抜き、再度差し込む。

(ホットプラグの実験)

再度差し込んだ時に windows が再度認識し、その後通常のキーボードとして使用可能。

何度も繰り返しても使用可能であった。

- ・ 通常動作中、インタフェースノードをネットワークから切断する(UTP ケーブルを引き抜く)

ネットワークから切断している間は(当然のことながら)キーボードのデータは伝わらなかったが OS による認識は続いていた。再度ネットワークへ接続(UTP ケーブルを刺し直す)するとデータの転送が再開する。

これは、接続された時点でホスト側インタフェースが各種ディスクリプタを獲得/保存し以降は自身が USB デバイスになりすましてホストとのコミュニケーションを行うために可能となる。デバイス側からの信号がない場合は現状を維持する。

再接続後、デバイス側からの信号が来た場合にはそのまま切断前の環境が再現される。

7 デバイス・ハブを展開するには

7-1 対応バス・ポート

今回、デバイス・ハブの実装の一例として USB での実装を行った。前出の通り USB は近年ローカルバスのなかで中心的な位置を占め様々な用途に幅広く用いられているが、しかしながらデバイス・ハブを展開するためにはその他のバスへの対応も必要となる。以下にその例を挙げる。

ユーザインタフェース用ポート

- ・ PS/2
- ・ DVI
- ・ ADB

ストレージ用デバイス

- ・ IDE
- ・ SCSI

汎用バス

- ・ Firewire
- ・ シリアルポート
- ・ パラレルポート
- ・ IrDA

7-2 マッチングポリシー

ホスト - デバイスのマッチングの際に用いられるポリシーについて以下にいくつか提案する。

7-2-1 空間をキーとしたマッチング

例えば、書斎に複数台のコンピュータが存在する場合、それらに対してデバイスを切り替えて使うことが可能となる。

もう少し広く空間をとり家中と考えた際には、その中の家電や調理器具、AV 機器など(これらはホスト側)がマッチングの対象となる。それらで1つのデバイスを切り替えて用いることで例えば既存のキーボードが「何でもリモコン」と呼べるような複合的なヒューマンインタフェースデバイスへと変貌する。また、ストレージを前述のような機器で共有することによって自分のいる場所でオーディオをならしたり、調理器具のコントロールや様々な機器のログ情報等を集約することが出来る。

空間のスコープはマッチングサーバにプログラムしておき、適宜切り替えることで適切かつ効率的なマッチングを行うことが可能である。

例えば、机上 < 室内 < 屋内 < 敷地内 という形でスコープを定義する。このとき、ターゲットスコープを机上とすることで作業スペース内に限定したマッチングを行い、また部屋の AV 機器やエアコンなどを操作しようとする場合はその部屋にスコープ合わせると言う形で使用する。

それぞれのスコープをドメインという形で定義することで、DNS を使用して実現できる。

同一空間へデバイスを持ち寄ってそれぞれが同一ホストにマッチングされることでそのホストの中のアプリケーションのユーザとなる。これによってリアルスペースを共有し F2F コミュニケーションを絡めたアプリケーションが実現される。これがもっとも生きるのはゲームの世界であると考え。ベイゴマやミニ 4 駆を持ち寄って遊んだ経験はないだろうか。ビデオゲームが普及し、その世界の中のキャラクタを自分のエージェントとしてしまうことが当たり前になってくるにつれ、この「自分(のエージェント)を持ち寄って遊ぶ」感覚は廃れていった。リアルスペースとバーチャルスペースの距離感・空気感。これら両者の新しいバランスを生み出すゲームがデバイス・ハブ環境で実現される。

7-2-2 所有者をキーとしたマッチング

人ごとに ID をもち、これらをキーとしてマッチングを行うことで、「自分の」ホストに「自分の」デバイスからアクセスすることが出来る。遠隔地においてないし、特にオフィスのような多数のホスト/デバイスが混在する場に置いて何よりも「自分の」ホスト/デバイスへアクセスしたいと思うことが多い。「自分の」ストレージ、「自分の」電話を使いたいときに即それらへのマッチングを行う。

この場合、ホスト/デバイスとも所有者は一人に限らない。1 台のホスト/デバイスの所有者(ニュアンス的には)は複数でも良い。その場合、その人数分の ID がそのホスト/デバイスに関連づけられる。

7-3 要求事項

7-3-1 セキュリティ

昨今、随所で無線 LAN が用いられてワイヤレスなネットワークへの接続が実現されている。これは PC をネットワークに接続するのに大変手軽であり、用いられる機材も安価なため急速に普及している。ホテルや空港ロビー、レストランなどではワイヤレスネットワークのサービスが提供されているところも多く、インターネットへの接続の遍在性、簡便性、匿名性を提供しモバイルユーザに多大な恩恵をもたらしている。一方でそのセキュリティ管理の難しさが明らかになり、官公庁では原則使用禁止になるなどの対応がとられている。WEP や MAC アドレス制限など技術的な解決は複数存在するが、それらは先に述べた簡便性を犠牲にすることとなる。

デバイス・ハブ環境についても同様にセキュリティの問題が存在するが、本論文ではセキュリティに関しては指摘にとどめる。

7-3-2 帯域制御

デバイス・ハブは他のネットワークコミュニケーションになるべく干渉しない物でなければならない。ホストとデバイスとのコミュニケーションにおけるデータのやりとりはやむを得ないにせよ、マッチングに際して、通信量を極力抑えることが望まれる。AppleTalk や NetBEUI 等のプロトコルではネットワークのトラフィックを増大させてしまうデメリットがあった。例えば AppleTalk ではホストはネットワーク内の他のホストやデバイスを探すためにネットワークに繰り返しクエリーを流し続けていた。これらのクエリーには全てのホスト/デバイスは必ず応答を返す仕組みなのですでに検出されているホスト/デバイスもそのたびごとにトラフィックを生み出し、ネットワークを無駄に埋め尽くしている。デバイス・ハブ環境ではこのような chattiness を回避しネットワークトラフィックを最小限に押さえると同時に、アドホックなネットワークにおいてサービス通知やサービスルックアップをタイムリーに行えなければならない。ネットワークにつながったデバイスは適切にネットワークの状態を把握するとともに自身の存在を通知する。そして、以降の問い合わせは提供サービスを知る必要があるときなど、必要な場合に限り行われなければならない。caching や back off の技術などを適切に用いてこれらの理想状態へ近づける。

8 まとめ

本研究においてまず現状を整理し、そこでデバイス・ハブ構想を提案した。

そこでデバイス・ハブの機能要件を大きく「マッチング」と「ホストーデバイス間通信」とにわけ、それぞれに検討しデバイス・ハブ環境を設計した。

またその環境下で動作するデバイスとして USB キーボードをデバイス・ハブ デバイス化する実装を行った。これは新しいデバイスを一から作り出すわけではなく、既存のデバイスへデバイス・ハブ環境へのインタフェースを追加するという形で行った。

デバイス・ハブ環境/デバイス・ハブ デバイスを評価することでこれらの実現可能性を実証した。ここでは

目指した

- ・ ネットワークノードのオートコンフィグレーション
- ・ デバイスとホストのマッチング
- ・ OS 標準ドライバによる動作

の実現を実証できた。

さらにここで新しいデバイスを一から作り出したのではなく、既存のシステムのデバイス・ハブ化の実現を見ることによって既存システムとの親和性を確認することが出来た。

今後の課題として

- ・ 様々な対応バス・ポートの設計
- ・ マッチングポリシーの設計
- ・ 平易なマッチングポリシーの追加・削除機構
- ・ セキュリティ
- ・ 帯域制御

といったことが提示された。これらに対してそれぞれ検討を加えていくことでデバイス・ハブ構想はさらに発展してゆくであろう。

謝辞

まずこの論文の親を引き受けてくださった kwkt さん、サブ親を引き受けてくださった pierre さんへ。

いつもエキサイティングな経験を与えてくれる、そしてデバイス・ハブ構想の元となるアイデアのきっかけを下さった岩井俊雄さんへ。

人生の大先輩、ishi さんへ。

私のマイコン熱を再燃させてくれた nn さんへ。

絵を描いて下さった treasure さんへ。

abstract の英訳を引き受けて下さった uma さんへ。

気分転換を担当してくれた nir さんへ。

肩をもんでくださった irino さんへ。

執筆中いじめっこからかまってくれた don さんへ。

ともに産みの苦しみを分かち合った同期卒論生のみなさまへ。

アドバイスを下さった icar のみなさまへ。

いままさに歌いかけて下さっている井上陽水さんへ。

そして先生方へ。

and to Steve Jobs...

thank you and good bye.

参考文献

- Webpages -

- [1]Z80 Simulation, <http://digilander.libero.it/fricciardi/> (2002)
- [2]PIC, <http://www.microchip.com/1010/index.htm> (2002)
- [3]ATMEL, <http://www.atmel.com/> (2002)
- [4]TINI, <http://www.ibutton.com/TINI/> (2002)
- [5]IETF, <http://www.ietf.org/> (2002)
- [6]ZeroConf, <http://www.zeroconf.org/> (2002)
- [7]multicastDNS, <http://www.multicastdns.org/> (2002)
- [8]DNS extensions , <http://ietf.org/html.charters/dnsexst-charter.html> (2002)
- [9]DNS service Discovery, <http://www.dns-sd.org/> (2002)
- [10]TRON 協会, <http://www.tron.org/> (2002)
- [11]iTRON, <http://www.ertl.jp/ITRON/home-j.html> (2002)
- [12]富士ゼロックス, http://www.fujixerox.co.jp/tangible_mouse/ (2002)
- [13]日立セミコンダクタ, <http://www.hitachisemiconductor.com/> (2002)
- [14]ナショセミ, <http://www.national.com/> (2002)
- [15]AMD, <http://www.amd.com/> (2002)
- [16]cypress <http://www.cypress>
- [17]USB.org, <http://www.usb.org/> (2002)
- [18]USB HID information, <http://www.usb.org/developers/hidpage/> (2002)
- [19]USB developers document, <http://www.usb.org/developers/docs.html> (2002)

- Internet-drafts -

- [20]IPv4 Address Conflict Detection(Internet-draft)
(<http://files.stuartcheshire.org/draft-cheshire-ipv4-acd.txt>)
- [21]Dynamic Configuration of IPv4 Link-Local Addresses(Internet-draft)
(<http://files.zeroconf.org/draft-ietf-zeroconf-ipv4-linklocal.txt>)

- Books -

- [22]TECH I No.8 「USB ハード&ソフト開発のすべて」 CQ 出版社