システムソフトウェアと
オペレーティング・システム　89－２０
シ　ス　テ　ム　評　価　2－２０
（２００２．２．１５）

組込み用 OS における入出力管理の一方式

萱嶋 志門†　　並木 美太郎†

†東京農工大学大学院工学研究科

情報家電，オーディオ，ビデオカメラなどの組込みシステムは，ネットワークインタフェースやオーディオインタフェースを持ち，USB，IEEE1394 経由で他のシステムと結ばれることがある．このようなシステムでは，デバイス入出力の際に，ネットワークプロトコルに代表される規模の大きいデバイス制御用ソフトウェアや MP3 デコードなどのマルチメディア処理を実現するソフトウェアが使用される．これらの規模の大きくなったソフトウェアを従来のスタイルで一から開発するのは難しくなってきたが，これらの入出力用ソフトウェアは，その一部を共通に使用できることがある．すなわち，入出力の際に用いる各部品のインタフェースを定め，仮想化を行なうことが，開発工数軽減につながる．しかし，組込みソフトウェアには，割込み応答性能やハードリアルタイム性などの時間制約，プロセッサやメモリなどのハードウェアリソースの制限がある．また，TCP/IP のプロトコルスタックやオーディオ再生などの処理では，各階層間でのバッファコピーなど，無駄なハードウェアリソースの消費を避ける必要がある．したがって，入出力制御では，時間制約に対応できハードウェアリソースを極力消費せずにデバイスを仮想化する必要がある．今回提案する方式では，時間的な制約に対応するためにデバイスアクセス時刻をアプリケーション（以下，AP）から指定する機構，各種プロトコル間でのバッファコピーを削減するためバッファ管理をカーネルで行うことで入出力管理を行う機構を実現した．一方，AP からはデバイスアクセス要求時に，デバイススタックを指定することでデバイス制御を行うドライバ群を選択することができる．

A Scheme of I/O Management for Embedded Operating Systems

Shimon Kayashima†　Mitaro Namiki†

†Graduate School of Engineering,
Tokyo University of Agriculture and Technology

Embedded systems such as intelligent home appliance, audio and video camera has network interface or audio interface, and are connected to other systems via USB or IEE1394. When I/O is occurred, large scale software such as network protocol and multimedia software is executed on such system. While software's size is too large to develop from scratch. A part of software for I/O can be used as a common compornent. That is, to determin and virtualize interfaces of I/O components reduces time for development. But there are various limitations, for example, performance of processing interruption, time limitation in hard-realtime systems, and hardware limitation such as memory and cpu. Furthermore, it is necessary to avoid wasteful consumption such as copying buffer between compornents in TCP/IP protocol stack or audio player. Therefore, execution time must be managed, and devices must be virtualized with low resource. In this paper, a method to specify device access time and a method to manage buffer to reduce buffer copy are described. Application can control device by specifying "device stack" with I/O request.

1 はじめに

近年，組込みシステムでも汎用的なデバイスが採用されることが増え，入出力の仮想化が必要となりつつある．従来の組込みシステムでは，デバイス自体に汎用性があるものが少く，入出力には割込み応答性，ハードリアルタイム性が要求されオーバヘッドなどのデメリットに比べ，仮想化するメリットが少かった．このことから，組込み向け オペレーティングシステム (以下，OS) の標準規格とも言える μ ITRON では，入出力を仮想化した仕様が存在しない [2]．現在，組込みシステムでは，周辺デバイスとして，データロガーやモータ制御デバイスなどのハードリアルタイム性が要求されるものから，ソフトウェア規模の大きくなるイーサや USB などのデバイス，動画，音声の処理が必要となるオーディオインタフェースまで幅広く使われている．本稿で述べる入出力管理システムでは，これらのデバイスを仮想化することを試みる．

既に入出力の仮想化を行っている Linux や Windows などパーソナルコンピュータ (以下，PC) 向け OS は，多様なデバイスに対するドライバ自体の再利用性やアプリケーションプログラム (以下，AP) の再利用性を完全に満たそうとするため，組込みシステムで用いるには，性能的な面で問題が生じる．特に組込みシステムの場合，出荷後のデバイスやソフトウェアの変更がほとんどない状況であり，PC と同様の仮想化は必要ない．それよりも，次の問題を解決する事が重要となる．

(1) 時間制約

デバイスには，割込み応答性を必要とするものや，一定時刻にデバイスアクセスをする必要のあるハードリアルタイム性を要求するものがある．PC 向け OS では，割込みを完全に仮想化しているので，割込み応答性が遅くなる傾向にある．組込みシステムでは，割込み応答性だけのためにハードウェア性能を引き上げる必要が生じる可能性がある．また，並行処理をしている他プロセスによって実行時間が変動する可能性があり，ハードリアルタイムシステムには適用しにくい．例えば，Linux などでは，何単位時刻以降に起床するという指定はできても，ある一定時刻後以内に起床ということは指定できない．

(2) ハードウェアリソースの制限

PC 向け OS では，各サブシステムや AP がそれぞれにバッファを持つために，バッファ間コピーや同一データを持つバッファの存在などの無駄が生じている．したがって，ターゲットハードウェアに厳しい制限がある組込みシステム用途には向かない．

このような問題点を解決しつつ入出力の仮想化を行なう必要がある．仮想化を行なうことで，ソフトウェアの再利用性を向上させ，開発期間を短縮することができる．そこで，本稿では，バッファを元にした入出力管理機構を提案し，組込み用 OS 『開聞』上に入出力管理機構を実装した結果を述べる．以下，第 2 章で本稿で提案する入出力管理機構の目標と方針を述べる．第 3 章で本入出力管理機構の概要と各機構の説明を述べ，第 4 章でアプリケーション，デバイスドライバ向けの API を，第 5 章で『開聞』への実装を述べる．

2 入出力管理機構の目標と方針

入出力を仮想化するにあたって，前章で述べた時間制約，ハードウェアリソースの制限に対応する必要がある．物理デバイスへアクセスするドライバでは特に時間制約が厳しくなる可能性がある．一方，中間処理として，TCP/IP プロトコルスタックのような処理を行なう場合は，ハードウェアリソースの制限を満たすことが難しくなる．このような違いがあるので，このデバイスドライバをこの二つに分類して扱う．本稿では，物理デバイスへアクセスを行なうものを簡易的にデバイスドライバと呼び，そうでないものをプロトコルドライバと呼ぶ．本章では，時間制約，ハードウェアリソースの制限といった問題に対してそれぞれどのように解決するのか方針を述べる．

2.1 時間制約

時間制約とには，割込み応答性，ハードリアルタイム性が挙げられる．アンチロックブレーキやデバイスの異常を検知するなどの処理は割込み応答性能が問われる．割込み応答時間の最悪値が遅いとそれに合せてハードウェア性能を引き上げる必要が生じる．組込みシステムでは，ハードウェアコストを重視するので，このような事態は避けたい．また，データロガーやモータ制御などの処理はある時刻にデバイスアクセスを行なう必要がある．例えば，数 msec 間隔でデータロギングしているデバイスであれば，アクセス時刻が数 msec ずれただけでデータが意味のないものになる．したがって必ず基準となる時刻から一定時間以内にデバイスアクセスすることを保証するハードリアルタイム性が必要となる．

割込み応答性は OS に依存するので,『開聞』への実装の章で述べる．ハードリアルタイム性は，指定した時刻に起床することを保証する必要がある．これをカーネルに実装した起床時刻を保証するアラームスケジューラを用いて解決する．データロガーなどは AP によって取得時刻が変わるので，AP からも起床時刻を指定できる必要がある．

また，オーディオ再生に代表される処理では，ソフトリアルタイム性が要求されるので，デッドラインをミスした時にエラー復帰処理を行なう必要や，極力デッドラインミスが生じないようスケジューリングをする必要がある．今回の実装で用いた『開聞』にはエラー処理を行うためのデッドラインスケジューラがあるのでそれに処理を委ねる．

2.2 ハードウェアリソースの制限

消費するハードウェアリソースは少なければ少ないほど良いのは PC でも同じであるが，組込みシステムの場合，製品コストに影響をおよぼすため，ハードウェアリソースに厳しい制限がある．リソースの消費はカーネルサイズなど OS 自体の問題である部分もあるが，ここでは，入出力時のバッファ，キャッシュに使用されるメモリ，不必要なデータコピーなどを問題として取り上げる．バッファについては，AP 側から入出力先/元の領域を指定することでドライバ側で領域を確保する必要をなくす．

図 1 処理の入れ替え

2.3 ハードウェア仮想化

組込みシステムでは，後からハードウェアを変更したりソフトウェアを追加することは少ない．したがって，ソフトウェア開発時に必要となるデバイスドライバ，ミドルウェアの類が全て把握できる．つまり，カーネルで処理するべき動的に変更される部分は，PC 向けソフトウェアに比べ少ない．仮想化は動的な部分，例えば，デバイスの占有状態やバッファ制御に必要な部分などだけを対象に行なう．このようにすることで，カーネル内で余計な仮想化処理を減らすことができる．一方，プロトコルスタックやマルチメディアデータのデコードなどのミドルウェア，デバイスドライバはどのようなデバイスに使用されるのか，どういう場面で使用されるのか，設計，実装時にはわからない．したがって，これらのソフトウェアを組み合わせて使用することを考えると，ミドルウェア，デバイスドライバのインタフェースを整える必要がある．インタフェースが整っていれば，図 1 の暗号化処理のようにドライバアクセス時に共通して使用できるものを共用することができる．

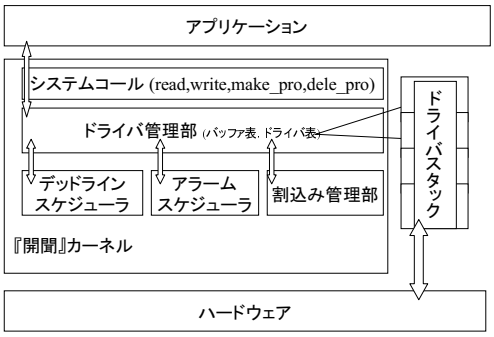

図 2 入出力機構の全体図

図 3 ドライバスタックの構成

3 入出力機構

3.1 本入出力機構の概要

入出力に必要となるのは，データ，デバイス，その間をつなぐ手続きである．本入出力管理機構では，これらをドライバスタック管理部を中心に制御する．全体を図 2 に示す．

本機構を用いると，AP は入出力要求をカーネル内のドライバスタック管理部に行なうことになる．本ドライバスタック管理部は，各ドライバをつなぐ役割を持つ．つまり，図 3 のような処理をつなぎ合せる役目をはたす．この流れをドライバスタックと呼ぶ．ドライバスタック内の処理はドライバスタック管理部により呼び出され，同期的に処理することを想定しており，逐次的に処理が行なわれる．アクセスドライバ呼び出し時には，物理デバイスとのデータアクセスに備え，割込みベクタやアラームスケジューラへアクセスドライバを登録する．また，デバイスアクセスが終了すると，AP から指定された方法で指定された先に入出力完了通知を送信する．

AP 側では，入出力の際に，データを格納するバッファ領域を確保し,『開聞』バッファを作成する．デバイスアクセス時までに，その属性を初期化する．また，どのドライバをどのように使用するのか，ドライバを使用する順序を決定し，作成したドライバスタックにドライバの実行順序を記入する．このような設計にしたのは，AP 設計者が，組込みシステムでは，後からソフトウェアやハードウェアを変更することが少ないので，AP を設計，実装する時に，使用するデバイスやデバイス制御の目的を全て知っていると考えられるからである．つまり，ドライバスタックは，AP 側で処理順番を決めることができる．AP 側で決めることで，柔軟なデバイス制御が行なえるようになり，ドライバ作成者は使用方法まで考えず単機能なドライバを作成すれば良いことになる．

AP では，ドライバスタックを決めた後，デバイスの排他制御のため，カーネルの持つ表にデバイスとドライバスタックの組を登録する．この表から各デバイスの状態を知ることができる．したがって，この表をドライバスタック管理部からアクセスドライバを呼び出す時に参照することで，デバイスの排他制御が行なえる．

以降の節で，各機構の詳細を述べる．

3.2 ドライバスタック管理部

ドライバスタック管理部では，AP 側で設定した個々のドライバスタックのデバイスとの対応や処理順番を参照しながらデバイスアクセス処理を制御する．具体的には，次の制御を行なう．

(1) ドライバスタックの処理を順番通りに処理する

(2) デバイスアクセスのタイミングをとる

(3) 入出力完了通知を行なう

次に処理の流れを述べる．AP が入出力要求命令を発行すると処理がドライバスタック管理部へ移行する．ドライバスタック管理部は，引数で与えられたプロトコル識別子を元に ドライバスタックを参照し，次に移行すべきプロトコルのエントリ関数を見つける．各プロトコルの処理が終了すると，ドライバスタック管理部へ処理が戻り，次の階層にあるプロトコルへ処理が移行する．最後のプロトコルまで到達し，その処理が終了すると『開聞』バッファにある入出力完了通知先へ終了を通知する．

非同期 I/O の場合，アクセスドライバは直接呼び出されず，デバイスが占有されていないか確認した後に，割込みベクタかアラームスケジューラへ登録される．この時，デバイスが占有されていれば，プロトコル管理タスクは，サスペンド状態になる．サスペンド状態の解除は，ドライバスタック管理部が行なう．同期 I/O の場合，他のドライバと同様に直接呼び出される．

また，ドライバスタック管理部では，デバイスドライバをアクセスドライバとプロトコルドライバの二つに分けて考えている．この二つでは，要求する性質が異なるからである．次に各プロトコルの性質と要求次項，それにどのように答えるのかを述べる．

3.2.1 アクセスドライバ

物理デバイスへアクセスするドライバである．想定している処理は，デバイスアクセスに要する命令だけを考えている．言い換えると，数命令から数百命令で終了する非常に短かい処理を想定している．問題となるのは，割込み応答性やハードリアルタイム性といった時間制約である．

割込み応答性能は OS に依存する．したがって，今回提案した方式だけでは割込み応答性能を向上させることはできないが，実装した『開聞』では，カーネルの機能をユーザが実装した機能に置き換えることで OS 使用前の割込み応答性に近い値を維持することができる．

一方，ハードリアルタイム性については，アクセスドライバを対象にスケジューリングすることで対応する．スケジューリングする時に必要となる，起床させる時刻や周期などは『開聞』バッファの属性にある．ドライバスタック管理部では,『開聞』バッファの属性値を参照してスケジューラへアクセスドライバを登録する．

3.2.2 プロトコルドライバ

直接物理デバイスへのアクセスは行なわないドライバを指す．基本的にタスクで行なわれる．データをバッファから読込み，加工し，書き戻すのが基本的な処理とする．このように仮定することで，各処理をつなげるのが容易になる．時間制約はアクセスドライバに比べゆるやかであるが，マルチメディアコンテンツの再生などソフトリアルタイム性が要求される．これにはデッドラインミスハンドラをスケジューリングすることで対応する．また，仮想化方式によってはサブシステム毎にバッファを持つことから発生する無駄がある可能性もある．今回はカーネル内でプロトコル間のバッファ管理を行なうことで，各ドライバがバッファを確保する必要性を省くことを試みた．

3.3 『開聞』バッファ

『開聞』バッファにはデバイスアクセスに必要となる属性を持つ.『開聞』バッファの属性を表 1 に示す．この属性は，デバイスアクセスに着目してつけた．大抵の場合，ドライバスタック中のドライバは，バッファアクセスが伴なう．このような属性を持たせれば，バッファアクセスの時に AP からの要求を読み取ることができる．

大抵の場合，AP からの要求後すぐにデータが得られる訳ではなく，その間，他タスクが実行される．この時のデータはバッファに蓄えられ，デバイスアクセスが終了すると AP へ渡される．また，ドライバスタックの各レイヤ間でのデータ送受信にも用いられる．データの送受信の際には，レイヤ間で用いるバッファが異なるとデータコピーのオーバヘッドが生じる．AP でそのバッファをキャッシュとして用いてるのでなければ，同一のバッファを使用でき，効率も良い．バッファを上書きして良いかどうかは AP で決定できる．

表 1 『開聞』バッファの属性

属性名	とりうる値
アクセス契機	周期，一回だけ (割込み，タイマ) 同期的に
入出力完了通知先	タスク識別子 0 でフラグを用いる
入出力完了通知方法	フラグ，強制起床
バッファの状態	使用中，未使用，上書き可能
データ要求サイズ	0 の時のふるまいはドライバによる
実際のバッファへのポインタ	
現在アクセス中のアドレス	

入出力完了通知は，カーネルの機能を用いる.『開聞』にはサスペンド/レジューム機能があるので，あらかじめ，入出力完了通知を行なうハンドラをサスペンドしておき，入出力終了にレジュームさせる方法や，広域変数などの領域にフラグをたてる方法などが考えられる．

3.4 アラームスケジューラ

データロガーやモータ制御などデバイスアクセスは時刻を指定して行なわれる場合もある．このようなデバイスではデータの入出力が行われる時刻がずれると意味をなさない．どれくらいずれると意味をなさないかはデバイスにより異なるが，ソフトウェアで対応できる精度として，数百μ秒くらいまでのずれまでは許容範囲と考えた．このようなアクセスハンドラを起床させるためにアラームスケジューラを実装した．アラームスケジューラは，タイマ割込みを仮想化したものである．

スケジューリング対象となるのは，アクセスドライバである．デバイスにアクセスするだけのハンドラなので数命令から数百命令で終了することを想定した．したがってアクセスドライバは，非常に短い処理と言える．割込み，ディスパッチを共に禁止にしておくことでコンテキスト退避，復元もハンドラで変更する可能性のあるものだけと最小限に抑えられる．このようにしてタイマ割込みからの応答時に極力処理を行わないことでタイマの精度を上げる．

表 2 入出力用 API

API 名	引数	処理内容
kmnIO	プロトコル識別子 バッファ識別子	プロトコル識別子で示されるデバイスからデータを取得する
makeproto	ドライバスタックへのポインタ	ドライバスタックを生成する

4 API/DDI

4.1 API

AP からの I/O 要求時には，次のパラメータを用いる．

(1) ドライバスタック識別子

(2) バッファ識別子

各パラメータ内のデータは，初期化しておく必要がある．また，ドライバによっては，他に必要となるパラメータがあるかもしれない．AP 実現時に使用するドライバは把握できるという前提で設計を行なっているので，そのようなパラメータについてどのように受渡しを行なうかは規定しない．ドライバ作成者が規定する必要がある．

4.2 デバイスドライバインタフェース

デバイスドライバ側から見たインタフェースを表に示す．データの入力元，出力先はカーネルが把握するので，デバイスドライバ側では自分の処理だけを行なえば良いことになる．したがって再利用性を各サブシステム毎に持たせることができる．各レイヤのエントリ関数は表 3 の引数を持つ．

参照先，参照元バッファへのポインタは，ドライバスタック管理部が自動的に割当てる.『開聞』バッファの属性値を参照したい時は引数の値を元に参照することになる.『開聞』バッファの属性は書込むこ

表 3 エントリ関数インタフェース

意味	
引数	参照元『開聞』バッファへのポインタ 拡張データへのポインタ
戻り値	成功：0 失敗：エラーコード

表 4 DD 向けインタフェース

kmnIOend	IO を終了して，ドライバスタック管理部へ処理を移行させる．普通に関数を終了させても良い
kmnAllowDispatch	ディスパッチ許可
kmnAllowInt	割込み許可

ともできるが，上書きすると元のデータがなくなるので，保存しておく必要がある．

拡張データは，階層間で独自に引数を持たせたい場合に用いる．例えば，ストレージデバイスへのアクセスには，ストレージ先のアドレスを知っている必要があるが，バッファとストレージ先の番地の対応はアクセスドライバでなく，ファイルシステムで実装されるかもしれない．このような時ファイルシステムからアクセスドライバへストレージ先の番地を送信する必要がある．ただし，この引数を用いると再利用性が落る．バッファサイズなどは,『開聞』バッファの属性値としてある．

アクセスドライバを割込みあるいはアラームで起床させる場合，割込み禁止，ディスパッチ禁止で実行される．タイムクリティカルな処理が終了し，データ加工などの処理をアクセスドライバ内で行ないたい場合，割込みやディスパッチを許可しておくことで，他タスクの割込み応答性，リアルタイム性への影響を最小限にすることができる．

4.3 入出力管理機構の使用例

入出力機構の処理例として，ストレージデバイスへのアクセスの流れを図 4 に示す．また，一番簡単な例として，データロガーを挙げる．このデータロガーは，データを取得するだけで加工は行なわない．また，指定された時刻に起床して，データを取得する．

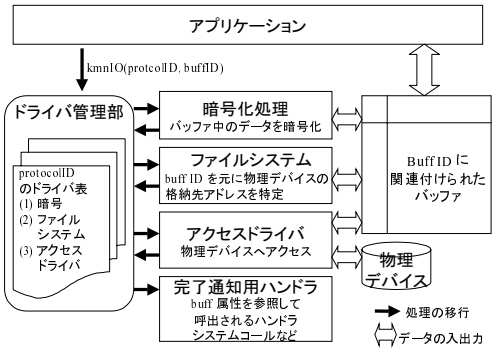

図 4 処理の流れの一例

(1) makeprot システムコールでプロトコルを作成

(2) バッファ領域を確保し，kbuff 構造体をセット

(3) kmnIO システムコールを用いて入出力を要求

(4) 非同期入出力要求ならディスパッチが生じずに処理は続行される

(5) ドライバスタック管理部に処理が移行し，ドライバが呼出される．

このドライバはロギングに必要となるレジスタを設定する．設定が終了すると処理をドライバスタック管理部へ戻す

(6) ドライバスタック管理部では，バッファ属性のアクセスタイミングを見てアラームスケジューラへアクセスドライバを登録する

(7) 起床時刻にアラームスケジューラによってアクセスドライバが起床する

(8) データを『開聞』バッファ属性にあるバッファ領域に格納し，同属性のカレントバッファポインタをすすめる

(9) 単位データを取得したら kmnIOEND ライブラリ関数を呼び出し，入出力完了を 『開聞』バッファの属性にしたがって通知する

(10) AP 側では，入出力完了通知を受けとったら，『開聞』バッファ属性で示されるバッファから値

を読み込む入出力完了通知を受けとる前でも，『開聞』バッファ属性で示されるバッファを参照すれば，データの読込みはできる．つまり，読み込みをさせながらデータを得ることもできる．

5 『開聞』への実装

5.1 『開聞』の概要

組込み用 OS 『開聞』は，アプリケーションの移植性とシステム全体の性能のトレードオフをユーザプログラマが選択できることを目標に開発がすすめられている．ターゲットとなっているプロセッサは，MIPS V_R4300，SH3/4，PowerPC403GA である．ボードはいずれも CQ 出版から発売されている評価用ボードである．

機能として，タスク管理，タスク間同期，割込み管理があるが，入出力の仮想化は行なわれていない．スケジューラは優先度ベースのものがあり，今回の実装で新たにアラームスケジューラ，ドライバスタック管理部が追加されている．さらに，デッドラインスケジューラが実装される予定である．

現在は割込み応答の部分，コンテキスト退避や割込み要因の特定処理をユーザプログラマの実装したハンドラに置き換えることができる．したがって，デバイスアクセス時に必要となる割込み応答性はこの機構をもちいることで OS を使用しない時に近い性能を実現できる．

5.2 ドライバスタック管理部

ドライバスタック管理部を実現する．入出力完了通知方法には，フラグあるいは指定のタスクを起床させることが選べる．『開聞』では，他タスクを起床させる手段としてレジュームシステムコールが用意されている．入出力完了通知方法として，他タスクを起床させる場合は，そのタスクをレジュームを用いて起床させる．ただし，オーバヘッドを考え，今回は起床させるだけなので，あらかじめ，起床させるタスクを生成しておく必要がある．

5.3 アラームスケジューラ

『開聞』には先に述べたように，優先度ベースのスケジューラが実装されている．それとは全く別にアラームスケジューラを実装した．アラームスケジューラはシステムコールでスケジューラにドライバを加える時とセットされたタイマから起床する時に呼ばれる．今回実装したアラームスケジューラは，次の動作をする．

(1) 要求された起床時刻順にドライバをキューに加える

(2) 先頭のキューに操作が加えられた時，先頭となったキューの起床時刻にタイマ割込みが生じるようにハードウェアタイマをセット，解除する

(3) タイマ割込みが生じたらキュー先頭のドライバを起床させる

6 おわりに

本論文では，入出力を組込みシステムで仮想化する一手法を提案し，筆者の属す研究室にて開発の行われている『開聞』上への実装を行なった．本方式では，AP 側から柔軟にデバイスアクセスを行なえることを示した．また，時間制約やハードウェアリソースの制限を満たせる機構を提案した．今後は本方式の評価と多くのデバイスへの対応を図る．

参考文献

[1] 萱嶋, 並木, 組込み用 OS『開聞』の割込み管理機構, 情報処理学会研究報告, 2000-OS-84, pp.47-54, 2000.

[2] 監修 坂村, μ ITRON 4.0 仕様, トロン協会 ITRON 部会, 1999.

[3] SH4 プログラミングマニュアル 第 4 版, 日立製作所, 2000.3.

[4] PowerPC 403GA User's Manual Second Edition, IBM 1995.3.