社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2005－ＳＬＤＭ－119（26）
2005／3／18

コンテンツプロテクションを目的とした カーネル内データパスの実現に向けて

追川 修一†

† 筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻
〒305-8573 茨城県つくば市天王台 1-1-1
E-mail: †shui@cs.tsukuba.ac.jp

あらまし デジタルマルチメディアデータは，複製し配信してもアナログデータのような品質劣化が生じないため，コンテンツが無断コピーされ配信されるようなことになると，コンテンツ権利者は甚大な被害を被ってしまう．そのため，暗号化に代表されるコンテンツ保護のための技術が開発されてきた．暗号化されたコンテンツを再生するためには，最終的には復号化されなければならない．復号化処理の解析を困難にするために，処理のハードウェア化，ソフトウェアの難読化といった技術が用いられてきた．しかしながら，ハードウェア化は処理が固定されてしまうため新しいテクノロジへの対応といった柔軟性に乏しく，またソフトウェアの難読化だけでは完全な耐タンパ性の実現は困難である．本研究では，保護を必要とするデータ処理を全てカーネル内で行い，プログラムやデータを不必要にユーザの目に晒さないようにすることで，耐タンパ性を強化することを目的とする．
キーワード オペレーティングシステム，組み込みシステム，セキュリティ，コンテンツ保護

Towards the Realization of In-Kernel Data Paths for the Purpose of Contents Protection

Shuichi OIKAWA†

† University of Tsukuba, Department of Computer Science
1-1-1 Tennodai, Tsukuba, Ibaraki 305-8573 Japan
E-mail: †shui@cs.tsukuba.ac.jp

Abstract Digital multimedia contents can be redistributed without quality degradation; thus, the loss caused by unauthorized duplication and use is significant for content holders. Data encryption has been mainly used to protect those contents. Contents data needs to be eventually decrypted to play them. In order to make decryption processing difficult to reveal, two major techniques, hardware implementation and software obfuscation, have been popularly used. While hardware implementation lacks the flexibility to deal with new processing methods, software obfuscation itself is not perfect solution for tamper resistance. This paper describes a software architecture that aims the reinforcement of tamper resistance by realizing data paths of digital multimedia contents within the kernel. Such in-kernel data paths prevent the unnecessary access to software programs and data that require protection.
Key words Operating Systems, Embedded Systems, Security, Contents Protection.

1. まえがき

近年，デジタルマルチメディアコンテンツの普及は目覚ましく，DVD のような ROM 媒体だけでなく，地上波及び衛星のデジタル放送やブロードバンドによるネットワーク配信も普及しつつある．また，高機能化が著しい携帯電話のようなポータブルデバイスへの配信も行われるようになってきている．

デジタルマルチメディアデータは，複製し配信してもアナログデータのような品質劣化が生じないため，コンテンツが無断コピーされ再配信されるようなことになると，コンテンツ権利者は甚大な被害を被ってしまう．そのため，暗号化に代表されるコンテンツ保護のための技術が開発，利用されてきた．暗号化されたコンテンツを再生するためには，最終的には復号化されなければならない．復号化処理の解析を困難にするために，

処理のハードウェア化，ソフトウェアの難読化といった技術が用いられてきた．しかしながら，ハードウェア化は処理が固定されてしまうため新しいフォーマットへの対応といった柔軟性に乏しい．一方，ソフトウェアの難読化は効率の問題を伴い，またそれだけでは完全な耐タンパ性の実現は困難である．

また，デジタル放送では高品位高精細のハイデフィニション映像（ハイビジョン）が実現され，非常に大きな帯域が必要とされるようになってきている．ハイデフィニション映像に親しんだユーザにとって従来のスタンダードデフィニション映像は非常に見劣りするものであるため，今後より多くのコンテンツでハイデフィニション化が進むと考えられる．データ量が増加するに従い，それを処理するシステムにも，より高い処理能力が求められる．組み込みシステムなどコストも重視されるシステムにおいては，より効率の良い処理方法を用いることでプロセッサの負荷を下げることができれば，より安価なシステム構成が可能になる．

本研究では，保護を必要とするデータ処理を全てカーネル内で行い，プログラムやデータを不必要にユーザの目に晒さないようにすることで，耐タンパ性を強化することを目的とする．また，データ処理を全てカーネル内で行い，データのコピーや移動，それに伴うキャッシュやTLB操作の低減，及び正確な資源管理により，処理の高効率化を目指す．ソフトウェアで処理を行うことの利点は，プログラムを更新することにより，新しいテクノロジへのアップグレードが容易であることにある．暗号化されたコンテンツの復号化とデコード処理が1チップ化されたようなハードウェアでは，使用できるテクノロジが固定されてしまう．長期間使用されることの多い組み込みシステムでは，安価ではあるが購入時点で使用できるテクノロジが固定されてしまうシステムと，やや高価ではあるがアップグレード可能なシステムの選択肢がある方が望ましい．ソフトウェアで処理を行うことにより，アップグレード可能なシステムの構築が容易になる．

2. 関連研究

カーネル内で組み合わせ可能なモジュールを提供し，モジュール間でのデータ転送機構を提供するものとしてSTREAMS [9]がある．STREAMSは，UNIX System Vのキャラクタ型デバイスドライバをモジュールするために設計された．様々な機能を持つSTREAMSモジュールを積み重ねることにより，必要な機能を持つデバイスドライバを構成することが可能である．データの流れは，ユーザプロセスからデバイスへのダウンストリームと，デバイスからユーザプロセスへのアップストリームに分けられる．ダウンストリーム，アップストリームそれぞれのために定められたモジュールインタフェースを呼び出すことにより，データ転送は行われる．STREAMSはネットワークプロトコルスタックの実装にも使われるほどの機能を持つが，基本的にデバイスドライバを構成するために設計されたため，インタフェースによってはブロックしてはいけないなどの制約があり，また非同期に動作可能にするために優先度や時間制約の概念を持たないため，正確なマルチメディアデータパスの実現には適さない面を持つ．

Splice [3]は，効率の良いマルチメディアデータ転送を可能を実現するため，ファイルディスクリプタを連結するシステムコールを導入した．連結されたファイルディスクリプタ間のデータ転送はカーネル内で行われる．Spliceは，基本的にはSTREAMSのユーザプロセス側の終端を結ぶ機能であり，STREAMSと同様に優先度や時間制約の概念を持たない．

マイクロカーネルアーキテクチャの研究により，データ転送は著しく効率化された．カーネル機能をマイクロカーネルとサーバに分割するマイクロカーネルアーキテクチャ[4], [5]では，データ転送の効率化が大きな問題であった．モノリシックカーネルでは単純なコピーにより実現されていたデータ転送方式では，マイクロカーネルアーキテクチャではコピーの回数が増加してしまうため，全体的が性能が低下してしまう．そのため，コピーをせずにデータ転送を行うために，仮想記憶機構を用いたページの転送や共有による効率化が考えられた．また，サービスをさらに細分化しデータ処理をローカルに行うことでデータ転送コストを低減する方法[6]も考案された．マイクロカーネルアーキテクチャでのデータ転送の効率化の対象となったのは，マイクロカーネル上で動作するサーバで扱われるデータである．本研究では，データをカーネル内に留めることにより効率化を実現するという点で，本質的に異なっている．

異なる保護ドメイン間でデータ転送を効率化する研究として，共有メモリを用いる方法では，ユーザプロセスとカーネルの間でのデータ転送を効率化する研究[2], [10]，RPCを効率化する研究[1]があり，ページ転送を用いる方法としてContainer Shipping [8]がある．これらを用いることにより，異なる保護ドメインで動作するプログラムでのデータ転送を効率化することは可能である．しかしながら，共有メモリまたはページ転送では，基本的にはデータを共有することにより効率化しているため，コンテンツ保護の観点からはデータへのアクセスをより容易にしているという点で望ましくない．

3. カーネル内データパス

本節では，まずカーネル内データパスの実現に向けて必要なデータパスの分離の概念について述べる．次に，そのデータパスをどのように駆動するかについて述べ，最後にSTB (Set Top Box)でのカーネル内データパスを用いたシステム構成例について述べる．

3.1 データパスの分離

本研究は，保護を必要とするデータ処理を全てカーネル内で行うことで，プログラムやデータを不必要にユーザの目に晒さないようにし，またデータ処理必要な資源を正確に管理することにより，処理の高効率化を目指すものである．そのためにカーネル内にデータパスを実現する方法をとるが，プログラム全てをそのままカーネル内に実現すべきではない．本研究では，保護すべきデータ転送を伴う処理のみをプログラムから分離してカーネル内に実現する方法を提案する．

図1は，リモートホストからのファイルコピーにおけるデータパスの分離の例を示している．図1左に示したFTPでは，

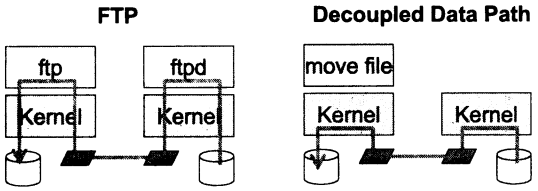

図 1　データパスの分離
Fig. 1　Decoupled Data Path

リモートホストからファイルを取得する場合，送信側は ftpd，受信側は ftp プログラムによってファイル転送のための処理が行われる．ftp，ftpd のそれぞれはユーザプロセスとして実行される．まず ftp がユーザからどのファイルを転送するかの要求を受け付け，その要求を ftpd に伝える．送信側である ftpd は要求されたファイルのデータを読み出し，ftp に転送する．読み出されたデータは一旦 ftpd を実行するユーザプロセス内に読み込まれてから，ftp とのネットワークコネクションに対し書き出される．受信側の ftp は，ftpd とのネットワークコネクションから転送されてきたデータを読み出し，ファイルに書き込む．受信側の ftp も ftpd と同様に，データはユーザプロセス内に一度読み込まれてから書き出される．これに対し，NFS によりリモートホストからファイルを取得するデータ転送の場合，送信側は全てカーネル内でデータが処理される．しかしながら，受信側のプログラム（例えば cp コマンド）では通常のファイル入出力として処理されるため，データはユーザプロセス内に一度読み込まれてからファイルに書き出される．

NFS を用いた場合，送信側ではデータ転送がカーネル内で行われるが，受信側でもカーネルで転送されてきたデータを直接ファイルに書き込むことにより，データをユーザプロセス内に読み込むことなくデータ転送を実現できる．その様子を図 1 右に示す．受信側でユーザプロセスとして実行されるプログラムはユーザからどのファイルを転送するかの要求を受け付け，その要求をカーネルに伝える．カーネル内のファイル転送機構は，その要求に従いリモートホストに対し要求を伝える．リモートホストのカーネルから転送されてきたデータはファイルに書き出されるが，データ転送は全てカーネル内で行われ，ユーザプロセスにはデータが読み込まれることはない．このようにデータパスを分離することにより，データ転送処理を全てカーネル内部で行うことが可能になる．

上述したように，データ転送を伴うプログラムは，通常データの転送元と転送先そしてそのデータにどのような処理を行うかといった要求をユーザから受け取り，その要求に従ってデータ転送及びそのデータの処理を行う．これまで，そのような処理を行うプログラムはユーザプロセスとして実行され，まずデータ読み取り要求をカーネルに対して行い，データをユーザプロセス内に読み込む．そして，そのデータに対して処理を行い，処理を行った後のデータを書き込む要求をカーネルに出すことで，例えばファイルや別のプロセスといった，必要な箇所にデータを転送する．図 2 にその様子を図示する．

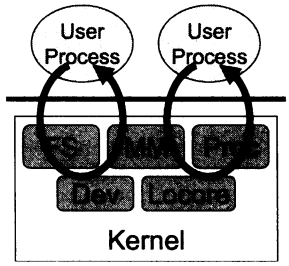

図 2　ユーザプロセスによるカーネル機能の呼び出し
Fig. 2　User Processes' calling Kernel Funcations

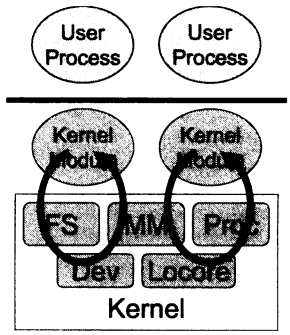

図 3　カーネル内モジュールによるカーネル機能の呼び出し
Fig. 3　In-Kernel Modules' calling Kernel Functions

同様のデータ転送は，カーネル内にデータに対して行う処理を実装したモジュールを置き，そのモジュールがカーネル内のデータ読み出し及び書き出し機能を呼び出すことでも実現できる．ユーザプロセスは，ユーザからの要求を受け取り，その要求に従ってユーザプロセスからそのカーネル内のモジュールに対して制御コマンドを発行する．図 3 にその様子を図示する．

ユーザプロセスの要求に応じたデータパスをカーネル内に作成し，そのデータ転送及びデータに対する処理を行うカーネル内モジュールを実現するための課題は，データパスの設定及び制御方法，カーネル内モジュールの実現方法の 2 つに大別される．データパスの設定及び制御方法については，

- 設定制御対象とするデータパスの指定
- データパスの構成モジュールの指定
- データパスの構成手順

といった問題点があり，またカーネル内モジュールの実現方法については，静的リンク又は実行時ロードして動的リンクするかというリンケージの問題がある．カーネル内モジュールを実行じデータパスに沿ってデータ転送及び処理を行う実行エンティティの問題については，次節で述べる．

3.1.1　設定制御対象とするデータパスの指定

データパスはアプリケーションの処理を言わば肩代わりするものであり，カーネル内に予め設定されているサービスではない．従って，アプリケーションが実行時に作成，設定，制御できるものでなければならない．また，データパスはアプリケーションが実行されるユーザプロセス固有ものである．アプリ

ケーションを構成する複数のユーザプロセス間での共有を許すとしても，データパスの基点及び終点，連結されるモジュールといったデータパスの構成モジュールは，作成されたデータパスごとに設定可能でなければならない．

このような要求を満たすためには，データパスはカーネル内に作成されるオブジェクトである必要がある．カーネルはデータパスを作成するシステムコールを提供し，そのシステムコールはデータパスの作成が成功した場合，ユーザプロセス内で一意に決まるIDを返す．ユーザプロセスは，そのデータパスIDを指定し，その後の設定，制御を行う．

3.1.2 データパスの構成モジュールの指定

データパスは，データパスの基点及び終点，連結されるモジュールから構成される．データパスを使用するユーザプロセスは，システムコールによりデータパスを作成しデータパスIDを取得した後に，そのデータパスを構成するモジュールを指定することで，データパスを完成させる．そのためには，データパスの構成モジュールを指定する方法と，データパスを完成させるための手順が決まっている必要がある．以下，データパスの構成モジュールを指定する方法を述べ，次にデータパスの構成手順について述べる．

データパスの構成モジュールを指定する方法として，起点と終点となるデバイスについては，デバイスファイルを用いることができる．デバイスファイルをオープンし，そのファイルディスクリプタをIDとして用いるようにすれば，そのデバイスに対するアクセス権，デバイスへのアクセスの排他制御はオープン時に検査することができる．データパスの途中でデータに対し処理を加えるモジュールに関しても，同様にデバイスと考えることができる．これらのモジュールはデバイスとしてのインタフェースを提供するが，実際のデバイスとは関連付けられていない．入力データに対し何らかの処理を加えて出力する仮想的なデバイスである．実際には起点と終点も，例えば常に同じデータを創出するデータジェネレータや又はデータをただ受け取るだけのヌルデバイスのような実際のデバイスと関係しない仮想的なデバイスとすることができるとテスト段階では非常に便利であるため，データパスの構成モジュールは全てデバイスとして扱うと，統一されたインタフェースで扱うことができる．

3.1.3 データパスの構成手順

データパスに構成モジュールを連結していく手順としては，STREAMSと同様に，LIFOの順序関係を持つスタックを用い，データパスの起点から終点の順序で構成モジュールをプッシュしていく．カーネルは，そのためのシステムコールを提供する．データパスIDとプッシュする構成モジュールのID，そして操作のアトリビュートを指定しシステムコールを呼ぶことで，構成モジュールが連結される．起点となる構成モジュールはデータを生成するものでなければならず，また終点となる構成モジュールは送られてきたデータを消費しデータを次に渡す構成モジュールが不必要なものでなければならない．また，連結される構成モジュール間で受け渡されるデータフォーマットもコンパチブルでなければならない．従って連結時には，指定されたアトリビュートと，プッシュされる構成モジュール，その前に来る構成モジュールから，連結が可能かどうかがチェックされる．連結の成否は，システムコールの戻り値から判別できる．

構成モジュールの連結が失敗した場合の回復手段として，すでに連結してしまった構成モジュールを取り除く操作も必要である．データパスの構成モジュールは，LIFOの順序関係を持つスタックとして扱われるため，最後に連結されている構成モジュールをポップして取り除くことができればよい．すでに連結された構成モジュールをポップすることにより必要なだけ取り除いた後，再び別のモジュールをプッシュすることができる．

3.1.4 カーネル内モジュールのリンケージ

データパスの構成モジュールとなるカーネル内モジュールは，何らかの方法でカーネルの一部として導入されなければならない．カーネルの一部としてカーネル機構を呼び出すためには，カーネルの一部としてリンクされる必要がある．リンケージの方法としては，カーネルのブートイメージの一部となるように予め静的リンクをしてしまう方法と，ブート後実行中のカーネルにモジュールを導入する動的リンクの方法がある．

静的リンクされたカーネルのブートイメージ，動的リンクされるカーネルモジュールのどちらも，不正なアクセスにさらされる危険性があり，不正に改変された機能によりコンテンツが盗まれるようなことがあってはならない．そのため，静的，動的どちらのリンク方法を用いても，コンテンツ保護をのためには，実行するモジュールに不正な改変が加えられていないかの検査機構は必要である．また，ROM又はファイルシステムに保存されるカーネルのブートイメージやモジュールについても，処理内容を保護するための暗号化等による保護も必要である．

3.2 データパスの駆動

データパスは，それを利用するユーザプロセスからは独立して自律的に動作するものである．従って，データパスに沿ってデータを移動し処理するための実行エンティティは，ユーザプロセスとは別に必要になる．最近の多くのオペレーティングシステムカーネルでは，カーネル内の様々な処理を独立して並行に行うためにカーネルスレッドを提供している．従って，データパスの駆動にはカーネルスレッドを用いる．

カーネルスレッドの割り当て方法としては，大別して以下の3つの方法が考えられる．

- 1つのカーネルスレッドで全てのデータパスを処理
- 各データパスにスレッドを割り当てる
- データパスの構成モジュールごとにスレッドを割り当てる

1番目の方法では，1つ（又はSMP対応にするためにはプロセッサ数と同じ数）のカーネルスレッドが複数のデータパスを流れるデータを処理する．この場合，複数あるデータパス上のデータのうちどれをカーネルスレッドは処理するのかというスケジューリングが必要になる．データパス又はデータに優先度を付けることにより，最も優先度の高いデータから処理することが可能になる．しかし，より高い優先度のデータが現れた場合，切り替えのタイミングはデータパスの構成モジュール間の転送時となり，画像コーデック処理などの処理時間が長い構成

モジュールを含む場合は，応答性が悪くなる可能性がある．また，カーネルスレッドにデータの優先度を反映するための処理も煩雑である．

2番目の方法では，1つのデータパスを流れるデータを処理するために，1つのカーネルスレッドが割り当てられる．割り当てられたスレッドにより，データパスに沿って次の構成モジュールへのデータの転送，構成モジュールでの処理が行われる．この場合，データの優先度を固定的にカーネルスレッドに割り当てることができる．そのため，1番目の方法では問題となったデータの優先度をカーネルスレッドに反映させるための煩雑な操作は不要である．また，データパスの構成モジュールがマルチスレッド対応である場合には，構成モジュールを処理中であってもより優先度の高いカーネルスレッドに切り替えることもできる．データパスとカーネルスレッドの1対1対応が取れるのであれば，資源予約[7]による安定した処理も可能である．問題点としては，データパスの分岐，結合が起こる場合に，データパスとカーネルスレッドの1対1対応を取るのが難しくなることがあげられる．このような場合には，分岐先に新たなカーネルスレッドを割り当てたり，結合点でデータを受け取る同期を取るといった構成が必要になる．また資源予約を行うためには，データパスを構成する全てのカーネルスレッドで使用する資源をまとめて予約する必要がある．

3番目の方法では，各構成モジュールがそこでの処理をするためのカーネルスレッドを作成する．カーネルスレッドをは受け取ったデータを処理し，そのデータが属するデータパスに従ってデータを転送する．ある時点で各構成モジュールが処理するデータは1つであるため，1番目の方法と同じく，応答性や優先度反映の問題が生じる．1番目の方法と異なるのは，各構成モジュールが並行して動作可能である点である．そのため，この方法はデータパスの構成モジュールごとに専用プロセッサを割り当てるようなモデルに対応しやすく，またデータパス途中の構成モジュールがハードウェアに置き換わった場合にも対応しやすいという利点を持つ．

3番目の方法の拡張として，各構成モジュールに複数のカーネルスレッド（ワーカスレッド）を割り当てる方式も考えられる．ワーカスレッドには処理をする優先度（グループ）が割り当てられ，より高い優先度のデータが到着した場合，現在処理中のワーカスレッドをからより高い優先度のデータを処理するワーカスレッドへ処理が切り替わる．これにより，応答性や優先度反映の問題を軽減することが可能である．ワーカスレッドを用いる場合の欠点は，スレッド数が多くなり，スレッド管理，スレッド間の同期のコストが大きくなってしまうことである．

本論文では，最も単純であると考えられる2番目の方法をとるものとする．しかしながら，3番目又は3番目の拡張の方法についても拡張性の面でメリットがあると考えられるため考慮を続ける．

3.3 STBにおけるデータパス例

本節では，STB (Set Top Box) でのカーネル内データパスを用いたシステム構成例について述べる．STBは元々ケーブルテレビのためのチューナであったが，デジタル衛星放送への対応

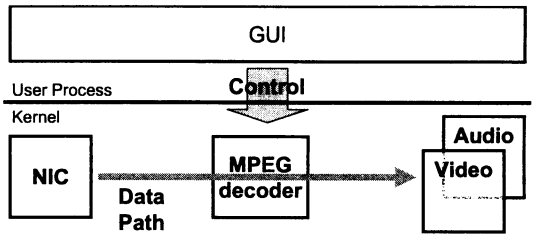

図4 データパス分離の例
Fig. 4 Example of a Decoupled Data Path

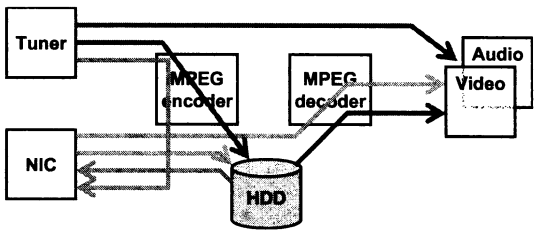

図5 STBにおけるデータパス
Fig. 5 Data Paths in STB

やケーブルテレビのデジタルテレビ化により多機能化がはじまり，ブロードバンドを通したオンディマンドテレビ放送 (Video on Demand) への対応，HDDへのコンテンツの保存といった機能も持つようになってきている．図4は，そのようなSTBのデータパス例を図示したものである．データパスの詳細については次に述べる．ユーザプロセスは近年肥大化が著しいGUIを司る．ユーザはGUIを通しデータパスを制御する．動作中のデータパスはGUIと並行して，分離されたカーネルスレッドにより処理されるため，必要なQoSを確保することが容易である．マルチメディア処理が必要なデータパスをGUIから分離することで，セキュリティ強化が必要なデータパスをよりシンプルにすることができ，GUI部分のソフトウェアのバグ等の脆弱性の影響を受け難くなると考えられる．

多機能化が進んだSTBは図5に示されるような構成モジュールとデータパスを持つものと考えられる．構成モジュールとしては，放送受信のためのチューナ，データのコーデック（エンコーダ，デコーダ），データ保存のためのHDD，映像，音声出力のためのデバイスがある．データパスとしては，これらを結ぶ様々なパスが考えられる．データパスの終点が表示出力デバイスであるとは限らないため，複数のデータパスが同時実行されることも考えられる．異なった要求を持つデータパスを並行して処理するためには，それぞれのデータパスに対する優先度及び時間管理，そしてデータ転送の効率化が重要であることがわかる．

4. プロトタイプ実装

現在，Linuxカーネルを用いてプロトタイプを実装中である．Linuxカーネルは，Vine Linux 3.1に含まれるLinux 2.4.27をベースにカーネル内データパスを実現する機構を追加して

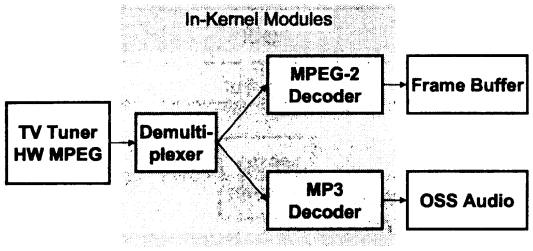

図 6 プロトタイプ実装概観
Fig. 6 Overview of Prototype Implementation

いるところである．データパスを分離するアプリケーション例としては，ビデオ再生ソフトウェア MPlayer(注1) をベースに，コーデックモジュールをカーネルモジュール化する作業を進めている．MPlayer は多種多様なコーデックや出力デバイスをサポートする多機能なビデオ再生ソフトウェアであるため，コーデックは MPEG2 に限定し，またビデオ出力デバイスとしては VESA ビデオフレームバッファ，音声出力デバイスとしては OSS (Open Sound System) を用いるよう，機能的に絞って作業を進めている．

図 6 にプロトタイプの概要を示す．テレビチューナからはビデオ及び音声の両方を含む MPEG2 データが送出されてくる．まずこのデータを MPEG2 ビデオデータ，MP3 音声データに分離する．次に，ビデオ，音声それぞれのデータのデコード処理を行い，VESA ビデオフレームバッファ，OSS 音声デバイスに出力する．ここで，テレビチューナドライバ，MPEG2 データの分離，MPEG2 ビデオデータデコーダ，MP3 音声データデコーダ，VESA ビデオフレームバッファドライバ OSS 音声デバイスドライバがデータパスの構成モジュールとなる．データパスは MPEG2 データの分離のために分岐するため，1 つのカーネルスレッドで全てのデータパスを実行できず，分岐先で新たなカーネルスレッドを必要とする．

MPEG2 ビデオデータデコーダ，MP3 音声データデコーダといったコーデックをカーネル内に実装する上で問題となるのは，浮動小数点演算である．通常プロセッサの FPU（浮動小数点演算ユニット）は，整数演算を行う CPU とは並行して動作する．そのため，システムコールの発行や割り込み等により CPU ではカーネルの実行が始まっていても，FPU はユーザプログラムのための演算を続けているということがありうる．カーネル内に実行が移るたびに FPU の演算終了を待ち状態を保存することは大きなコストを伴うため，これまで FPU を使用するような浮動小数点演算をカーネル内部で行うことはできなかった．しかしながら，近年カーネル内部でも多様な処理が要求されるようになり，例えば Linux ではソフトウェア RAID を実装するデバイスドライバではデータのブロックに対し高速な XOR 計算を行うために FPU を使用するようになってきている．そのため，必要に応じて FPU の状態を保存，回復している．

本プロトタイプ実装においても，FPU 使用時にはその状態を保存，回復する必要がある．しかしながら，今後マルチコア，マルチスレッディングのプロセッサが増加していくものと考えられ，十分な数の並行実行がサポートされるようになれば，データパスの実行でコア又はスレッドを占有する方法も考えられる．

(注1)：http://www.mplayerhq.hu/

5. ま と め

本論文では，保護を必要とするデータ処理を全てカーネル内で行い，ソフトウェアプログラムやデータを不必要にユーザの目に晒さないようにすることで，耐タンパ性を強化することを目的としたカーネル内データパスの実現に向けた設計，実装について述べた．カーネル内データパスにより，保護と必要とするデータ処理とは無関係な GUI 等のプログラムを分離することで，安全性をより高める．また，データ処理を全てカーネル内で行い，データのコピーや移動，それに伴うキャッシュや TLB 操作の低減，及び正確な資源管理により，処理の高効率化を目指している．

謝辞 本研究の一部は KDDI 研究所からの受諾研究により行われました．ここに感謝致します．

文 献

[1] B. Bershad, T. Anderson, E. Lazowska and H. Levy. Lightweight Remote Procedure Call. *ACM Transactions on Computer Systems*, 8 (1), pp. 37–55, 1990.

[2] P. Druschel, and L. L. Peterson. Fbufs: A High-Bandwidth Cross-Doman Transfer Facility. In *Proceedings of the 14th ACM Symposium on Operating System Principles*, pp. 189–202, 1993.

[3] K. Fall and J. Pasquale. Improving Continuous-Media Playback Performance with In-Kernel Data Paths. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pp. 100–109, June 1994.

[4] D. Golub, R. Dean, A. Forin and R. Rashid. Unix as an Application Program. In *Proceeding of the Usenix Summer Conference*, June 1990.

[5] J. Liedtke. On μ-Kernel Construction. In *Proceedings of the 15th ACM Symposium on Operating System Principles*, December 1995.

[6] C. Maeda and B. N. Bershad. Protocol Service Decomposition for High-Performance Networking. In *Proceedings of the 14th ACM Symposium on Operating System Principles*, pp. 244–255, December 1993.

[7] S. Oikawa and R. Rajkumar. Portable RK: A Portable Resource Kernel for Guaranteed and Enforced Timing Behavior. In *Proceedings of IEEE Real Time Technology and Applications Symposium*, pp. 111–120, June 1999.

[8] J. Pasquale, E. Anderson, and P. K. Muller. Container Shipping: Operating System Support for I/O-Intensive Applications. *IEEE Computer*, 27 (3), pp. 84–93, 1994.

[9] D. M. Ritchie. A Stream Input-Output System. *AT&T Bell Laboratories Technical Journal*, 63 (8), pp. 311–324, 1984.

[10] M. N. Thadani and Y. A. Khalidi. An Efficient Zero-Copy I/O Framework for UNIX. Technical Report SMLI TR-95-39, Sun Microsystems Laboratories, 1995.