タイムスタンプ付ストリーム I/O による音の実時間処理

栗田 亮† 千葉 滋† 光来 健一‡

†東京工業大学情報理工学研究科数理計算科学専攻
Dept. of Mathematical and Computer Sciences, Tokyo Institute of Technology

‡NTT 未来ねっと研究所
NTT Network Innovation Laboratories

kurita@csg.is.titech.ac.jp

我々は Linux において、音のアナログ録音再生に関しての実時間処理を行うシステム TS-I/O(TimeStamp-I/O) を提案する。従来のリアルタイム OS は音の実時間処理にはあまりむいていなく複雑である。一方、我々の TS-I/O を使うと I/O 割り込み時にタイムスタンプを取得することができる。このタイムスタンプを調べることでアプリケーションは正常に録音再生できたかを知ることができ、容易に最低限のデータ品質を保証することができる。

1 はじめに

自動伴奏システムをご存知だろうか。このシステムは、ユーザーをソリストと見立て、ユーザーのメロディーに対してそれに合わせて伴奏を返すシステムである。このシステムでは、入出力のストリームデータをリアルタイムに処理することが必要である。

しかし、従来の OS ではこのようなストリームデータを正常に処理できない場合がある。これは、与えられた時間内に処理を行う実時間処理機能が欠けていることに原因がある。これに対し、現在リアルタイム OS が開発されているが、これらの OS はストリームデータの処理にはあまりむいていない。

そこで我々は、Linux カーネルの追加システムとして TS-I/O(TimeStamp-I/O) を提案する。TS-I/O を使うと I/O 割り込み時にタイムスタンプを取得することができる。このタイムスタンプを利用することで、時間軸に沿った音の同期再生をすることが可能であり、サウンドデータの品質を最低限保証することができる。

本論文では、以下、2 章で Linux の音の処理にどのような問題があるかを述べ、3 章で我々が提案するシステム TS-I/O に関して説明する。4 章では TS-I/O を追加したことによるオーバーヘッドを示し、5 章ではタイムスタンプを利用した既存の技術について紹介する。最後に 6 章でまとめと今後の課題を述べる。

2 音の実時間処理

この章では汎用 OS である Linux での音の処理について、どのように行われているか、どのような問題があるかを説明する。また、この問題に対処するために開発されているリアルタイム OS について、その特徴と問題点を説明する。

2.1 音の録音再生

音の録音再生は、図 1 のようにハードウェア〜カーネル〜アプリケーションの 3 つの間でのデータのやりとりによって行われる。ADC はアナログ・デジタル変換器、DAC はデジタル・アナログ変換器である。

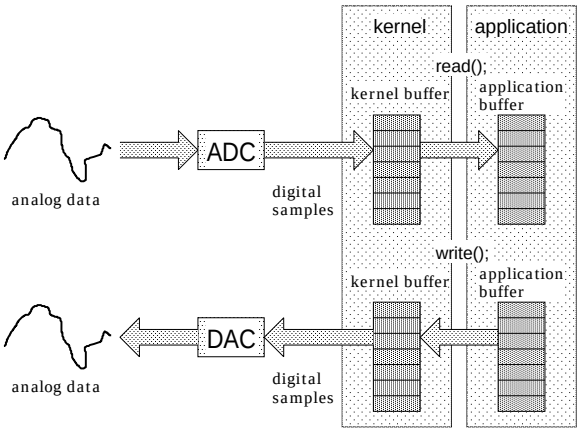

図 1: 音の録音再生でのバッファリング

録音の場合、サンプルデータは ADC によりデジタル変換され、カーネルバッファにバッファリングされる。カーネルはアプリケーションにそのデータをバッファリングすることで録音は終了する。再生の場合は録音の逆の流れである。

録音再生中のミスの原因はハードウェア〜カーネル間のデータ損失と、カーネル〜アプリケーション間のデータ転送の遅れの２つがある。この２つの中で音の処理では特に多い、カーネル〜アプリケーション間の問題に注目した。このデータ転送の遅れとはハードウェア〜カーネル間の転送に対して、カーネル〜アプリケーション間の転送が遅いということである。この遅れによって次の２つの現象が生じる。

1. **音飛びの発生:** 録音時に生じる。録音データが一部抜け落ちたものになってしまう。カーネルバッファが一杯になり、溢れたデータが破棄されてしまうのが原因。
2. **間の発生:** 再生時に生じる。再生が途中でしばらく止まってしまう。カーネルバッファが空っぽになり、再生するデータがなくなってしまうのが原因。

このようなカーネル〜アプリケーション間の問題は、OS に大きな負荷がかかっている場合に生じる。音の処理と同時に大きな負荷をかける処理を行っていると、このような処理のミスが生じてしまう。

2.2 実時間処理

データ転送でのミスは、従来の汎用 OS が実時間処理機能を提供できていないことに原因がある [1]。実時間処理とは、デバイスからの入力信号やプログラムからの要求に対して与えられた時間内にこれを処理し、処理できない場合はその後の処理に影響を及ぼさないように対処することである。実時間処理にはこのような即時性と周期性の正確さが必要とされるが、メディアストリームの処理においては特に周期性の正確さが重要となる [2]。例えば動画の再生中に、あるフレームの処理が遅れてしまった場合、それ以降のフレームを時間軸にそって再生できるように一部のフレームの再生を飛ばしても問題はない。

従来の Linux にはプログラムの優先度を指定できるシステムが備わっている。これにより重要な処理の優先度を上げることで実時間処理が可能に思える。

しかし、これだけでは実時間処理をすることはできない。Linux は元々実時間処理に対応したアーキテクチャではなく、優先度の高い処理が、優先度の低い処理やハードウェア割り込みなどによって妨げられる場合がある。また、Linux には処理ミスが生じたかどうかを検知する機能が無いので、このミスに対処することができない。

2.3 リアルタイム OS

現在、実時間処理を行うためのリアルタイム OS が開発されている。リアルタイム OS は元々、組み込み分野などの専用 OS として開発されてきたが、近年では汎用 OS としてのリアルタイム OS も開発が進められている。RT-Linux[3] や ART-Linux[1] などが挙げられる。これらのリアルタイム OS はタスクスケジューリングを厳しく管理することで、与えられた信号や要求を必ず時間内に処理するシステムである。

しかし、これらのリアルタイム OS はメディア処理にはあまりむいていない。その理由の１つに、リアルタイム OS が行うような時間に厳しい処理はメディア処理では必要とされていないことが挙げられる。たとえ途中の処理に失敗したとしても、その後のデータが時間軸に沿って処理できれば処理全体としては問題はないと考えられている [4]。しかし、従来のリアルタイム OS は処理を必ず終わらせることを目的に作られており、万が一処理に失敗したときの場合の処理についてはあまり考えられていない。リアルタイム OS がメディア処理に向いていないもう１つの理由は、従来のリアルタイム OS はユーザに大きな負担を強いる構造になっている点である。リアルタイム OS を利用するには複雑なプログラムをユーザーが用意しなければならない。また、リアルタイム OS が処理を安定して行えるようにするには、高性能なマシンを用意しなければならない。

3 TS-I/O

3.1 TS-I/O による音の実時間処理

我々が提案する TS-I/O(Time Stamp - I/O) は、入出力 (I/O) にタイムスタンプを利用するシステムである。この TS-I/O を利用することで、音のアナログ録音再生をリアルタイムに行うことができる。

TS-I/O を使うことで、録音が正常に行われたかどうか知ることができる。カーネルは録音でのバッファリング時にハードウェアからの入力データとともにタイムスタンプを記録し、アプリケーションに入力

データとそのタイムスタンプを渡す。アプリケーションはこのタイムスタンプを見ることで、データの損失があったか知ることができる。タイムスタンプの間隔に比べて読み出せたデータサイズが小さい場合、バッファリング時にデータの損失があったとわかる。

また、TS-I/O を利用して、時間軸にそった同期再生をすることが可能である。録音処理で得られたタイムスタンプを録音されたデータと一緒にカーネルに渡すことで、カーネルはタイムスタンプに従ってデータ処理を行うので、録音時に音飛びが発生していても時間軸にそった同期再生をすることができる。再生中に、アプリケーション〜カーネル間のバッファリングでのデータ損失が発生しても、同様に同期再生が可能である。

TS-I/O はカーネルの追加システムであり、タイムスタンプの付加やタイムスタンプを利用した同期処理は全てカーネルが行う。アプリケーションはデータと一緒にタイムスタンプの受け渡しをするだけで実時間処理を行うことができる。

TS-I/O が提供する実時間処理とは、2.2 節で述べた「処理できない場合はその後の処理に影響を及ぼさないように対処すること」である。TS-I/O のタイムスタンプを利用することで、処理ミスの検知と対処が可能である。従来の Linux には処理できない場合の機能は備わっていなかったが、TS-I/O が提供するこの機能を追加することで、汎用 OS である Linux でもストリームデータの処理で必要とされている実時間処理機能を備えることができる。

3.2 ネットワークにおける TS-I/O

TS-I/O は現在、ローカルでの音のアナログ録音再生の実時間処理機能を提供する。これを応用して、ネットワークパケットなどにも TS-I/O を適応できるように拡張することを視野に入れている。

TS-I/O を利用して、受信側は同期再生やパケットの損失の検知をすることができる。送信側はパケットにタイムスタンプを付加し、受信側はそのタイムスタンプを見ることでパケットの損失があったかどうかを知ることができる。また、このタイムスタンプに従って再生を行うことで、同期再生が可能である。

また、パケットを受信する時間を制御することもできる。指定された時間になるまでは受信側のアプリケーションに read させないようにすることができる。

3.3 TS-I/O の利点

TS-I/O の一番のポイントは、シンプルな設計である。Linux に対して、既存のリアルタイム OS のようにシステム全体について実時間処理機能を施したのではなく、I/O 部分にのみ実時間処理機能を追加した。これにより Linux の汎用性をそのまま利用でき、アプリケーション側は実時間処理のプログラミングを容易に行うことができる。また、そのシンプルな設計から安定性は高く、マシンの性能に依存することなく実時間処理が可能である。

次に、TS-I/O はカーネルレベルでの設計なので、アプリケーションレベルではできなかった実時間処理もすることができる。例えばローカルでの音のアナログ録音処理に関しては、OS 内でバッファリングされているせいでアプリケーションレベルでは検知できなかった音飛びを検知することができる。ネットワークでのパケット処理に関しては、受信側のアプリケーションの処理を制御することができる。

そして最後に、Linux 内で使われている全てのストリームで TS-I/O を利用できることが挙げられる。Linux を含め Unix では、ファイルだろうとネットワークだろうとみなストリームに抽象化されている。全てのストリームにタイムスタンプをつけられるようになれば、時間的制約のある処理を行うアプリケーションのプログラミングが非常に楽になる。

3.4 実装

以上述べてきた音のアナログ録音再生処理に関しての TS-I/O を実装した。対象とした OS は linux-2.2.19 で、サウンドカード Ensoniq AudioPCI (ES1371) のデバイスドライバ es1371.c に機能を追加した。入力の割り込み処理時にタイムスタンプを記録する。また、再生処理時にタイムスタンプに従ってカーネルからハードウェアへデータ転送を行う。

このシステムを利用するために、追加システムコール read_gettbuf()、write_puttbuf() を用意した。このシステムコールを read()、write() の代わりに呼ぶことで、タイムスタンプの受け渡しをすることができる。引数として、従来の read()、write() の引数に加えてアプリケーション側のタイムスタンプ用のバッファとそのサイズを必要とする。read_gettbuf() を呼ぶとこのバッファにタイムスタンプが格納される。write_puttbuf() ではこのバッファを渡すことで、時間軸にそった同期再生をすることができる。

現在、各デバイスドライバに機能を追加する設計になっているが、今後は汎用的なルーチンを作り、それを呼ぶことでデバイスドライバを変更できるようにする予定である。

4 実験

TS-I/O のオーバーヘッドとして、割り込み関数、read 関数、write 関数のオーバーヘッドを測定した。デバイスドライバの割り込み関数内ではタイムスタンプの付加処理を追加し、read 関数と write 関数ではそのタイムスタンプに対する処理を追加した。

実験に使った計算機は Pentium III 733MHz, メモリ 512MB のものである。処理にかかったクロックを測定して、その平均と誤差を求めた。

割り込み関数のオーバーヘッドは平均 $2.3 \pm 0.5\mu$sec という結果になった。まれに原因不明だが 120μsec ほど時間がかかることがあった。この時間だけデータの損失があるかと思われるが、ハードウェア〜カーネル間の転送は CPU を使わずに行われるのでこの転送が追加処理によって妨げられることはない。実際、割り込み関数が呼ばれる時間に対するデータサイズを調べたところ、割り込み間隔に誤差が多少ある場合もあるが正常なデータサイズを取得できた。

また、read 関数、write 関数のオーバーヘッドは各システムコールの処理時間 500μsec に対し、数 μsec という結果になった。この時間だけカーネル〜アプリケーションの転送は遅れてしまうが、この程度なら関数自体にかかる時間の方が長いのであまり問題にならない。

5 関連研究

ここではネットワークでの、ストリームデータの実時間処理を行う既存の技術を紹介する。これらはアプリケーション側が用意する機能であり、TS-I/O のようにカーネル内部の機能とは異なるものである。

5.1 MPEG

MPEG は映像や音声の圧縮符号化方式である。符号化のアルゴリズムは現在幾つか存在し、扱うデータや通信ごとに使い分けられる。有名なものとして MP3 (MPEG-1 Layer III)[6] が挙げられる。

MPEG のシステムでは、各データパケットにタイムスタンプを含むヘッダーを付加する。このタイムスタンプに従うことで同期再生が可能である [5]。

5.2 RTP, RTSP

RTP (Real-time Transport Protocol) は映像や音声などのストリーミングのためのプロトコルである。RTP ではデータの先頭にタイムスタンプなどを含むヘッダーを付加して送ることで、メディア間の同期やパケットの損失の検出を行うことができる [7]。

RTP を利用したプロトコルとして、RTSP (Real Time Streaming Protocol) がある。RTSP はデータストリームの制御を目的とし、実際のデータ転送は RTP が行う。RTSP は具体的にはデータの早送り、巻き戻し、再生、停止などを行うことができる [8]。

これらのプロトコルは現在 Real Player や QuickTime などで使われている。

6 まとめと今後の課題

Linux において、音の録音再生に関しての実時間処理を行うシステム TS-I/O を提案した。I/O 割り込み時に取得したタイムスタンプを利用することで、実時間処理をすることが可能である。

今後は音以外のメディアデータ、例えば動画に関して TS-I/O を適応させるようにしたい。また、ネットワークパケットの転送でも TS-I/O を利用できるように研究を進めたい。

参考文献

[1] 石綿 陽一, リアルタイム処理を実現する ART-Linux の設計と実装, インターフェース, Vol.25, No.11, 1999, pp. 65–85.

[2] 河内谷 清久仁, 徳田 英幸, 「QOS チケット」モデルに基づく連続メディア処理の動的 QOS 制御, 第 7 回コンピュータシステム・シンポジウム論文集, 1995, pp. 141–148.

[3] M. Barabanov, A Linux-based Real-Time Operating System, 1997.

[4] A. Jones and A. Hopper, Handling Audio and Video Streams in a Distributed Environment, Proc. 14th. ACM Symposium on Operating Systems Principles, 1993, pp. 231–243.

[5] ISO/IEC 11172-1 International Standard MPEG-1 Systems, 1993.

[6] ISO/IEC 11172-3 International Standard MPEG-1 Audio, 1993.

[7] H. Schulzrinne and S. Casner and R. Frederick and V. Jacobson, RTP: A Transport Protocol for Real-Time Applications, 1996.

[8] H. Schulzrinne and A. Rao and R. Lanphier, Real Time Streaming Protocol (RTSP), 1998.