uITRON のカーネルモジュール化による組込み用 UNIX への融合

佐藤 喬　　多田 好克
{satou-t,tada}@is.uec.ac.jp
電気通信大学 大学院情報システム学研究科

概要

組込み機器の高機能化が進み，汎用 OS である UNIX を利用する事例が増えてきた．UNIX 上で動作している豊富なソフトウェア資産を流用することで開発コストを削減できる利点があるためである．しかし，UNIX には組込み機器を制御する上で重要視される実時間性が無い．そこで本研究では，実時間性を持つ uITRON をカーネルモジュール化し，UNIX へ融合する手法を提案する．UNIX と uITRON を融合することで，豊富なソフトウェア資産を持ち，実時間性のあるシステムが利用できるようになる．

1　導入

情報家電をはじめとした組込み機器の高機能化に伴い，組込み機器制御用のオペレーティングシステム (OS) として UNIX が使用されている．これは， uITRON のような既存の組込み用 OS に不足していたデバイスドライバやネットワーク機構，アプリケーションプログラムなどの豊富なソフトウェア資産をそのまま UNIX から流用できる利点があるためである．

UNIX が使用される例として，Web ブラウザ経由でネットワークに接続された組込み機器を制御することが挙げられる．このような場合，ネットワークを利用するために，UNIX が持つネットワークカードのデバイスドライバや TCP/IP のプロトコルスタック，Web サーバなどのソフトウェア資産を流用することが可能である．UNIX のソフトウェア資産を流用することで，ソフトウェア開発のコストを削減できる．

しかし，UNIX は汎用計算機用として発展してきたため，組込み機器制御に重要視される実時間処理に弱いという欠点がある．そこで本研究では，UNIX に実時間 OS の機能を組み込む手法を提案する．具体的には実時間 OS として，uITRON 準拠の実装である TOPPERS[1] をカーネルモジュール化し，LKM(Loadable Kernel Module) により UNIX 実装である NetBSD[2] へ動的に組み込む．NetBSD と TOPPERS を選択した理由は，両実装とも多様な CPU アーキテクチャに対応しているためである．組込み機器は目的に応じて様々な CPU アーキテクチャが使用されることから，対応する CPU アーキテクチャが多いことは利点となる．

本システムでの実時間処理は，uITRON のタスクとして記述でき，実時間性が必要ない処理は，UNIX のプロセスとして記述できる．このように，実時間処理を実現しつつ，UNIX が持つ豊富なソフトウェア資産を流用できることが本システムの利点である．また，uITRON の機能を LKM を利用し UNIX に融合しているため，uITRON の API(Application Program Interface) と UNIX のカーネル関数を単一プ

ログラム内で同時に扱うこともできる．

本システムは，豊富なソフトウェア資産と実時間性を要求する組込みシステムをターゲットとする．これらターゲットの要求を満たすため，豊富なソフトウェア資産を持つ UNIX と実時間性を持つ uITRON を融合する手法を取った．本論文では uITRON をカーネルモジュール化し，UNIX へ取り込む手法について議論する．

2　本システムの概要

本システムがターゲットとするのは，豊富なソフトウェア資産と実時間性を同時に要求する組込みシステムである．このような組込みシステムに対して，本研究では汎用 OS である UNIX に実時間 OS である uITRON をカーネルモジュール化して組み込むことにより解決する．

本システムの概要を Web 経由の UI で操作を行う監視ロボットへの要求を通して説明する (図 1)．

Web 経由で操作を操作を行うためには，監視ロボットを制御するシステムソフトウェア上で Web サーバが動作しなければならない．また，http を使った通信をするため TCP/IP のプロトコルスタックやネットワークデバイスのドライバが必要となる．さらに映像配信などのサービスを追加する場合には，対応したソフトウェアを必要とする．このようなソフトウェアは実時間 OS である uITRON では用意されていないため，開発のコストが高い．一方，汎用 OS である UNIX では，上記に述べたようなソフトウェアを豊富に持っており，それらを流用することで開発コストを抑えることができる．

しかしながら，監視ロボットの姿勢制御やセンサデータ読込といった実時間性を要求する処理に汎用 OS である UNIX は不向きである．UNIX は汎用 OS 向けに開発され，実時間性を持たないためである．そこで，本システムでは実時間性をもつ uITRON をカーネルモジュール化し，UNIX へ取り込む．このようにすることで，豊富なソフトウェア資産を持つ UNIX と実時間性を持つ uITRON を一つのシステムとして用いることができる．

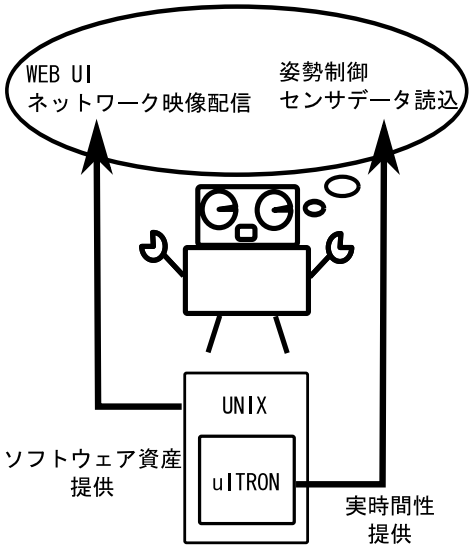

図 1　本システムの概要

3　設計

本システムの設計方針は以下の３つである．

- 多様な CPU アーキテクチャへの対応
- UNIX，uITRON への変更を極力抑える
- uITRON の実時間処理を確保

これらの方針を通して，本システムの設計について述べる．

3.1 CPU アーキテクチャへの対応

まず，多様な CPU アーキテクチャへの対応について説明する．組込みシステムはその用途に応じて様々な CPU が使われており，多様な CPU アーキテクチャに対応することは，本システムの適用できる範囲を広げることにつながる．このことを実現するには，使用する UNIX と uITRON 自身が多様な CPU アーキテクチャに対応していることが望ましい．

そこで，本システムで使用する UNIX と uITRON の実装は，それぞれ NetBSD と TOPPERS を選択した．選択の理由は，これらの実装が豊富な CPU アーキテクチャに対応しているためである．両実装の特徴として，CPU アーキテクチャ依存部と非依存部のが明確に切り分けられており，他 CPU アーキテクチャへの移植が容易である利点もある．

3.2 UNIX と uITRON への変更

次に，UNIX，uITRON への変更を極力抑えることについて説明する．使用する UNIX と uITRON の実装である NetBSD と TOPPERS はオープンソースとして開発が続けられている．そのため，それぞれのソースを直接編集し大幅な変更を加えると，新しいバージョンへ追随することが難しくなる．

そこで，本システムでは NetBSD と TOPPERS について次のように対応する．NetBSD に対する変更は LKM(Loadable Kernel Module) と呼ばれる動的にカーネルに組み込むカーネルモジュール機構を利用する．LKM を使うことで，NetBSD のカーネルソースへは変更を加える必要が無くなり，新しいバージョンへの追随が容易になる．ただし，作成した LKM プログラムに新しいバージョンへの依存部が存在した場合は修正が必要である．また，LKM では実現が難しいカーネル関数の置換などの修正についてはカーネルソースへの変更は避けられない．ただし，その修正箇所は極力抑えるよう努める．

TOPPERS に対する変更は直接ソース変更を行うしか方法が無い．そのため，NetBSD の LKM 側で吸収できる変更はそちらで行い，TOPPERS に対する変更の量を抑える．また，TOPPERS に対する変更は，NetBSD の LKM を一つの計算機アーキテクチャとして考え，アーキテクチャ依存部として記述する．このことにより，TOPPERS のアーキテクチャ非依存部への変更を無くし，修正箇所を減らすよう努める．

3.3 uITRON の実時間性確保

最後に，uITRON の実時間処理を確保する事について説明する．NetBSD へ TOPPERS を組み込むにあたり，TOPPERS の実時間性を損なわない事は重要な課題である．この課題を達成するためには，以下 2 つの問題を解決する必要がある．

- TOPPERS に対する低遅延の割込み配送
- NetBSD による割込み禁止処理への対応

1 つめの問題は，割込み処理に関する問題である．タイマや TOPPERS が制御する周辺機器からの割込みが発生した場合，できる限り低遅延で TOPPERS へ処理を渡さなければならない．そのため，実時間性の無い NetBSD の割込み処理を介して TOPPERS へ割込み配送する事は避けるべきである．

そこで図 2 に示すように，割込み配送機構を LKM により実装し問題の解決を図る．割込み配送機構は，CPU が呼び出す割込み処理が記述された割込みベクタを置き換え，割込み処理をすべき NetBSD か TOPPERS へ割込み内容を配送する．このようにする事で，実時間性の無い NetBSD を介さずとも TOPPERS は低遅延の割込み処理ができる．

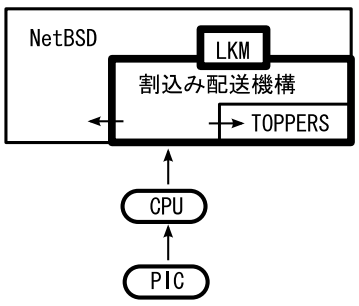

図 2　割込み処理の流れ

割込み配送機構は NetBSD と TOPPERS のコンテクスト切り替えも実現する必要がある．割込みの種類に応じて，もし TOPPERS が処理すべき割込みならば，TOPPERS へコンテクストを切り替え，そうでなければ NetBSD へコンテクストを切り替え処理を渡す．例えば，タイマ割込みを 1ms 周期に発生させ，基本は TOPPERS へ処理を渡し，10ms に 1 回 NetBSD へ処理を渡す，という事が挙げられる．保存すべきコンテクストは，命令ポインタやスタックポインタ，CPU レジスタなど必要最小限にとどめ，コンテクスト切り替えのオーバヘッドを抑える．なお，これに関して我々は et/MRSA[3] として FreeBSD 上への構築経験を有している．

2 つめの問題は，NetBSD による割込み禁止処理への対応についてである．NetBSD 側で CPU からの割込みを禁止されてしまうと，割込み配送機構へ割込み処理が届かなくなってしまう．もしこの状態の時に，TOPPERS 側でタイムクリティカルな処理を行いたいとしても，割込みがブロックされているため，処理を切り替えることができない．このことは，実時間性を損ねる原因となる．

NetBSD 内で割込み禁止をする大部分は splhigh() というカーネル関数を利用している．そこで，本システムではこの splhigh() を修正し，この問題の解決を図る．修正内容は，CPU からの割込みは禁止せず，割込み配送機構までは割込みを到達させるというものである．そして，配送機構からは NetBSD へ割込みを通知しないようにする．こうすることで，TOPPERS へは割込みの通知が可能になるため，TOPPERS の実時間性を損ねてしまうことを避けられる．

4　現状と今後の予定

現在，対象 CPU アーキテクチャを Intel x86 として実装を進めている．これは Intel x86 アーキテクチャを筆者が最も良く把握していることが理由である．現実装の完成後，PowerPC や MIPS など他 CPU アーキテクチャへ移植を試みる．

使用する NetBSD と TOPPERS のバージョンは NetBSD-5.0.1 と TOPPERS/JSP カーネル Release 1.4.3 である．このバージョンの TOPPERS は Intel x86 アーキテクチャをサポートしていないため，サポートに含まれていた TOPPERS/JSP カーネル Release 1.3 より移植した．

今後の予定として，完成した本システムを用いて，どの程度の実時間性を確保できるのかを評価する．割込み配送機構と NetBSD が動作することにより，TOPPERS が単体で動作するよりも実時間性が低下することが予想され

る．この実時間性の低下を測定し，実用に耐えることができるか考察する．

また，本システム上で動作するアプリケーションプログラムの作成を行う．本システムの特徴として，1 つのプログラム中から uITRON の API と UNIX のカーネル関数を使う事が可能であることが挙げられる．この特徴を用いたアプリケーションプログラムを作成し，本システムの活用について考察する．

5 関連研究

関連研究を大きく 3 つに分けて議論する．1 つめは，機能追加型である．これは汎用 OS に実時間機能を追加したものとなる．2 つめは，ハイブリッド OS である．これは汎用 OS を実時間 OS の 1 タスクと位置づけ動作させる．3 つめは，仮想計算機を用いるものである．これは物理計算機上に 2 つの仮想計算機を立ち上げ，それぞれで汎用 OS と実時間 OS を動作させる．

5.1 機能追加型

RTLinux[4] は Linux に実時間処理タスクの実行機能を追加する．タスクに対する割込み処理は通常の Linux カーネル内部の処理を通さないため，細粒度のハードリアルタイムを実現している．このハードリアルタイムを維持するため，タスク中からはカーネル内関数などを利用することはできず，独自のライブラリを用いたプログラミングが必要とされる．また，デバイスドライバを含むすべての割込み処理書き換える必要がある．そのため，ソースコードのないデバイスドライバは利用できない．

KURTLinux[5] は Linux で動作するユーザプロセスに実時間性を与えるものである．通常周期的なハードウェアのタイマ割込みのタイミングをユーザプロセスの時間要求に応じて変更する．このようにして，周期的なタイマ割込みで得られる実時間性よりも細粒度な実時間性を実現する．ユーザプロセスを使ったアプリケーション開発が可能という利点を持つが，Linux カーネルの介在が存在するため，中粒度のソフトリアルタイムまでが処理の対象となる．

et/MRSA[3] は FreeBSD で動作するカーネル外スレッド機構である．et/MRSA は独自のコンテクストを持ち，カーネル内のタイムアウト関数を使用することで，FreeBSD のカーネルコンテクストと et/MRSA のコンテクストを切り替えながら動作する．et/MRSA は LKM によりカーネルモジュール化されており，プログラム中からカーネル内の関数や変数にアクセスできる．ただし，コンテクスト切り替えがタイムアウト関数処理に依存しているため，実時間処理の粒度は中粒度にとどまっている．

5.2 ハイブリッド OS

Linux on TOPPERS[6] は TOPPERS の 1 タスクとして Linux を実行するものである．豊富なソフトウェア資産を Linux から流用し，実時間処理は TOPPERS のタスクとして記述できる．ただし，Linux 内部で割込み禁止処理をした場合，その間の実時間性が損なわれる可能性がある．また，1 つのプログラム中から同時に Linux の機能と TOPPERS の機能を呼び出すことに対応していない．2 つの OS 間で連携するためには，通信やデータ共有を用いる必要がある．

5.3 仮想計算機

MobiVMM[7] は実時間処理に対応した仮想計算機である．仮想計算機の代表格である Xen[8] は実時間処理に対応しておらず，割込みの配送のオーバヘッドが大きい．そこで，MobiVMM ではいくつかの I/O モデルを提案し，実時間 OS が動作する仮想計算機の割込み配送オーバヘッドを削減した．ただし，ハイブリッド OS と同様に 1 つのプログラム中から汎用 OS と実時間 OS の機能を同時に利用することはできない．

6 まとめ

情報家電などの組込み機器の高性能化に伴い，豊富なソフトウェア資産を利用でき，かつ，実時間性を持ったシステムソフトウェアが求められている．そこで本論文では UNIX とカーネルモジュール化した uITRON を融合することを提案した．

UNIX は汎用 OS として発展しており，デバイスドライバやネットワーク機構，アプリケーションプログラムなどの豊富なソフトウェア資産を持っている．一方，uITRON は優れた実時間性を持っており，両者を融合することでそれぞれの特徴を兼ね備えたシステムソフトウェアとなる．

参考文献

[1] TOPPERS プロジェクト: http://www.toppers.jp/

[2] The NetBSD Project: http://www.netbsd.org/

[3] 多田 好克, 福田 伸彦，鈴鹿 倫之，中村 嘉志: “カーネルの外部で走行するカーネルスレッドの提案とその実装法”, 電子情報通信学会論文誌, vol. J85-D-I, no. 3, pp. 286–293, (2002).

[4] Michael Barabanov and Victor Yodaiken: “Introducing Real-Time Linux,” *Linux Journal*, no. 34, pp. 19–23 (1997).

[5] Balaji Srinivasan, Shyamalan Pather, Robert Hill, Furquan Ansari and Douglas Niehaus: “A Firm Real-Time System Implementation Using Commercial Off-the-Shelf Hardware and Free Software,” *Proceedings of the Fourth IEEE Real-Time Technology and Applications Symposium*, pp. 112–119 (1998).

[6] 保田 信長, 飯山 真一, 冨山 宏之, 高田 広章, 中島 浩: “Linux と ITRON によるハイブリッド OS の設計と実装”, 情報処理学会研究報告 (SLDM), pp. 45–50, (2004).

[7] Seehwan Yoo, Miri Park and Chuck Yoo: “A Step to Support Real-Time in Virtual Machine,” *6th Annual IEEE Consumer Communications and Networking Conference*, pp. 1–7 (2009).

[8] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt and Andrew Warfield: “Xen and the art of virtualization,” *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 164–177, (2003).