4N－1

リアルタイムシステムにおける通信手法 (1)
— リアルタイム OS への STREAMS 機能拡張 —

中西 茂利，古賀 太，山口 義一
三菱電機 (株) 情報技術総合研究所

1 はじめに

近年，マルチメディア分野などへの適用のため，分散システムの各ノード間での高リアルタイム通信が重要視されている。そのためには，OS 内の通信用デバイスドライバの設計実装手法を検討する必要がある。

筆者らは，STREAMS 機構 [1] を拡張することで，高リアルタイムのデバイスドライバ設計手法を考案した。インタフェースは従来と互換のまま，これまで実現されていなかった，デバイスドライバ，および同系で動作する他のスレッド (プロセスを含む) との間の細かな優先度制御を可能とした。

2 リアルタイムスケジューリング

リアルタイム OS とは，“系内で明示された応答時間制限を満たさなければならないと共に，障害も含めた苛酷な条件に対応しなければならない” [2] とされている。OS がどのような状態にあったとしても，その上で動作しているスレッドは，決められた応答時間制限以内に確実に実行されなければならない。この応答時間制限を満足するため，リアルタイム OS では，固定優先度スケジューリングに代表される，優先度制御という手法を用いている。高優先度が付与されたスレッドほど，優先的に CPU 時間が割り当てられ，その結果，応答時間を短縮出来ることになる。

また，OS 内の資源確保 / 解放に関しても，応答時間制限を満たすよう，優先度制御の手法を元に設計されている。例えば，低優先度のスレッド (以下，T_L) が OS 内の資源使用権を確保後，高優先度のスレッド (以下，T_H) が同一資源を確保しようとした場合，優先度の逆転現象が発生してしまう。これを回避するために，一時的に T_L の優先度を T_H のものに遷移させ，T_L が確保している資源を早期に解放させるよう，スケジューリングを OS に促す，“優先度継承機能” を使用しているリアルタイム OS も存在する。これにより，優先度の逆転現象が発生せずに円滑な資源確保 / 解放が可能となり，高リアルタイム性実現に寄与している。

3 STREAMS 機構の問題点

図 1 に STREAMS 機構により構築されたデバイスドライバの動作例の一部を示す。

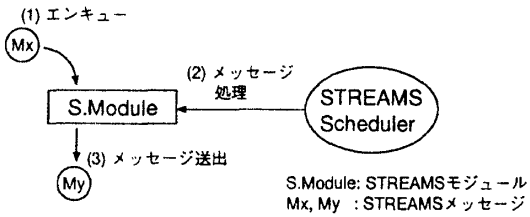

図 1: デバイスドライバ動作例

図 1 内の S.Module にはキューが備わっており，STREAMS 構成要素間での STREAMS メッセージ (以下，単にメッセージと呼ぶ) の双方向通信，および蓄積に使用される。一般的な処理の流れは次のようになる。

(1) エンキュー：他の STREAMS 構成要素から受信した M_x を，S.Module の読み込みキューにキューイングする。

(2) メッセージ処理：コンテキストスイッチ発生時に動作した STREAMS スケジューラによって，S.Module で規定されたメッセージ加工処理ルーチンが実行されることで，M_x から M_y が作成される。なお，STREAMS スケジューラはカーネル内コードとして実装されている。

(3) メッセージの送出：M_y を他の STREAMS 構成要素へ送出する。

なお，複数のメッセージを S.Module が受信した場合は，付与されているメッセージ優先度を元に処理順序を決定し，上記 (1) ～ (3) を実行することになる。

しかし，本手法をリアルタイム OS に適用しようとした場合，次の問題が発生する。まず，メッセージ優先度はメッセージ処理順序の決定のみに使用されており，優先度制御には使用されていない。そのため，高メッセージ優先度を付与しても応答時間を

A communication method for a realtime system (1).
— A STREAMS extention of a realtime OS —
Shigetoshi Nakanishi, Futoshi Koga, Yoshikazu Yamaguchi
Mitsubishi Electric Corp. Information Technology R&D Center.

短縮出来ないことがある。次に，STREAMS スケジューラはコンテキストスイッチ発生時にメッセージ処理を実施するため，あるメッセージ処理を開始すると，それに費やす時間の長短を問わず，終了までプリエンプション不可能となる。これらの問題より，構築されたデバイスドライバは，優先度制御手法を阻害してしまい，その結果応答時間制限を逸脱してしまう現象が発生することになる。

4 STREAMS 機構のリアルタイム拡張

筆者らは，第2章で示したスケジューリング手法を有するリアルタイム OS 下での，STREAMS 機構のリアルタイム拡張を実施した。

4.1 リアルタイム拡張実装時の動作

図2に本拡張時のデバイスドライバの動作例を示す。

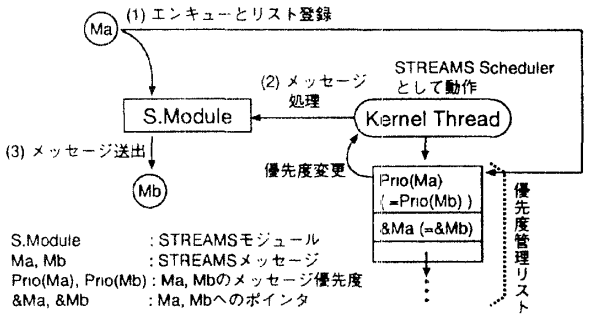

図 2: STREAMS 拡張実装方式による動作例

図2中のカーネルスレッドは，カーネル空間で動作するスレッドであり，図1中の STREAMS スケジューラの機能を有している。本例では処理の流れは次のようになる。

(1) エンキューとリスト登録：M_a のエンキュー時に，prio(M_a) と &M_a を優先度管理リストへ登録する。なお，その際，メッセージ優先度により，優先度管理リストの各要素は降順にソートして登録する。これにより，優先度管理リストには，そのデバイスドライバ内に存在するメッセージのうち，メッセージ優先度の一番高いものが常時先頭にあることになる。

さらに，カーネルスレッドの優先度より prio(M_a) が高かった場合，prio(M_a) をカーネルスレッドの優先度として動的に変更する。

(2) メッセージ処理：prio(M_a) への優先度の変更に連動して，スケジューリングが実施される。カーネルスレッドが実行されると，優先度管理リストの先頭を参照し，処理すべきメッセージが M_a であることを検出する。その上で，M_a から M_b へのデータ加工処理が，prio(M_a) で実行される。なお，カーネルスレッドで実装していることにより，データ加工処理中でもプリエンプション可能である。

(3) メッセージの送出：M_b を他の STREAMS 構成要素へ送出すると同時に，M_b の処理がデバイスドライバ中で継続されるか否かを検出する。継続される場合は優先度管理リストから，M_b(=M_a) の情報に関する要素は削除しない。逆に，M_b の作成がデバイスドライバの末端の処理の場合，M_b の情報を削除する。これにより，各 STREAMS 構成要素での (1) ～ (3) の連続処理で発生する，優先度管理リスト中のメッセージ情報の冗長な登録 / 削除処理を回避する。

4.2 メッセージ処理の優先度継承

図2中のカーネルスレッドが，低優先度でメッセージ (以下，M_L) の処理を実行中に，高メッセージ優先度が付与されたメッセージ (以下，M_H) が書き込まれる場合が考えられる。このような，OS の内部的な優先度の逆転が発生した場合，本拡張手法では，第2章で述べた優先度継承機能を適用することで解決する。M_L を処理中のカーネルスレッドの優先度を，M_H のメッセージ優先度へ一時的に遷移させ，M_L の処理を早急に終了させる。その上で，M_H の処理を実施することで，円滑なメッセージ処理による応答時間短縮が可能となり，高リアルタイム性の実現に寄与することになる。

5 おわりに

本稿では，リアルタイム OS への STREAMS 機能拡張について提案した。本方式を用いることで，インタフェースは従来と互換のまま，これまで実現されていなかったデバイスドライバ，および同系で動作する他のスレッドとの間の細かな優先度制御を可能とした。本手法により，デバイスドライバでの応答時間短縮が可能となり，高リアルタイム性実現に寄与している。

今後，本手法を実際にリアルタイム OS 上で構築し，性能評価を実施する予定である。

参考文献

[1] Stephen A.Rago : "UNIX System V Network Programming", Addison-Wesley, 1993.

[2] 井原 廣一 : "1. リアルタイムシステムとは", 情報処理 Vol 35 No.1 pp.12–17, Jan 1994.