C-029

CPU リソース配分による車載インフォテインメントシステムの QoS 改善法

QoS Improvement Method for In-Vehicle Infotainment Systems by CPU Resource Allocation

服部 匠‡　　俞 明連‡
Takumi Hattori　Myungryun Yoo

1. はじめに

近年の自動車業界では，自動車の状態に関する情報(車両ステータス)や，映像や音声などのエンターテインメントの両方が提供できる車載インフォテインメントシステムが求められている．また，自動車の技術革新に伴い，ECU 間の通信に用いられる車載ネットワーク の配線が複雑化したことで，車両重量の増加や製造コストの増大などが課題となり，車載ネットワークと ECU の統合が進められている[1].

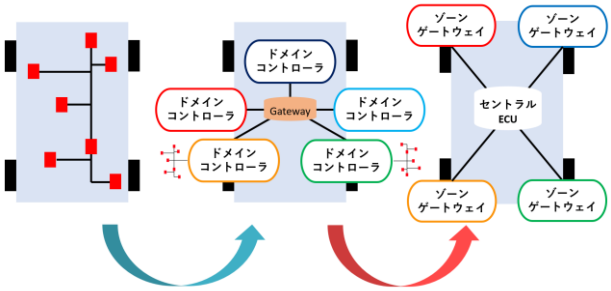

図 1　車載ネットワーク統合のイメージ

図 1 は車載ネットワークの統合の流れを示している．現在は多くの組込みコンピュータ同士が，用途に合わせて CAN(Controller Area Network) や LIN(Local Interconnect Network)，MOST(Media Oriented Systems Transport)などのネットワークで接続されているが，まず，いくつかの技術領域(ドメイン)ごとに集約され，最終的には車両の要所(ゾーン)ごとに集約される．

そのため，同一の組込みコンピュータ上で，制御系・情報系の融合した通信に対応し，リアルタイム性が保証できる大容量の通信ができるシステムが求められている．

ところが，このようなシステムを，マルチコアを有するコンピュータ上で汎用 OS を使用して構築した場合，制御系通信の QoS に影響を及ぼす可能性がある．

そこで，本研究ではこの課題の解決を目指し，車載インフォテインメントシステムを再現して影響を調査するために予備実験を行った．次に，その結果を踏まえて，組込みコンピュータ上の CPU リソースの配分によって QoS を改善する手法を提案し，先述した実験環境でその有効性を評価した．本論文ではその予備実験と提案手法について述べる．

‡東京都市大学　Tokyo City University

2. リアルタイム性における課題の分析

2.1 対象システムの構成

本研究が対象とする，車載インフォテインメントシステムを再現したシステムの全体構成を図 2 に示す．

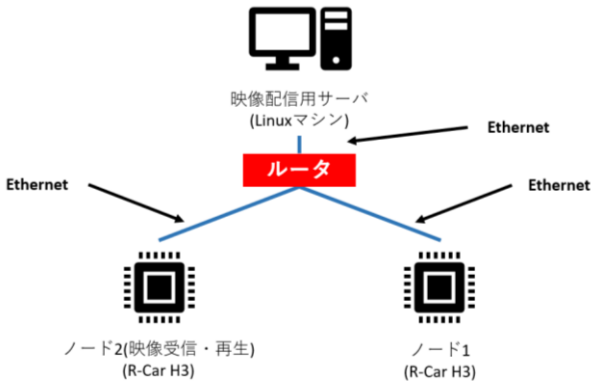

図 2　対象システムの全体構成

このシステムは，自動車の車内で車両ステータスと映像データを扱うことを想定しており，ルネサス社の R-CarH3 SoC を搭載した車載システム向け組み込みコンピュータ 2 台(以下，ノード 1, ノード 2)と映像配信用サーバから構成されている．これらのコンピュータはルータを介して Ethernet によって接続されており，ノード 1 とノード 2 の間では各ノード上で動作する RTOS(Real-Time OS)エミュレータによって車両ステータスの送受信が行われ，映像配信用サーバからノード 2 には映像データが送信される．RTOS エミュレータは我々がこれまでに開発したものを用いており，OSEK OS[2]のタスク管理機能をエミュレートして Linux 上でスレッドを動作して実現したものである．

2.2 QoS への影響評価実験

2.1 で述べた対象システムを実際に動作させ，映像データを流した際に，車両ステータスの通信に及ぶ影響を評価するための実験を行った．具体的には，ノード 1, 2 の間で車両ステータスを模したメッセージ A(8byte), メッセージ B(500byte), メッセージ C(1472byte)の 3 種類のメッセージを送受信させながら，同時に映像配信用サーバから 640×480(pixel)の映像を配信し，ノード 2 上で受信・再生させ，車両ステータスのやり取りにどのような影響が出るか確認した．その結果を表 1 に示す．

表 1 の結果から，対象システムで映像の受信・再生を行うと車両ステータスの往復時間が大幅に増大することがわかり，その平均値は 2[msec]を超えることが確認できた．

また，同時にノード 2 上で CPU コアの動作状況を確認したところ，コンテクストスイッチの回数が動画再生開始直後より大幅に増加していることが確認できた．

表 1　メッセージ往復時間の測定結果

実験条件	スレッド	平均値 (μ sec)	最大値 (μ sec)
映像の受信・再生なし	スレッド 0	332.28	725.51
	スレッド 1	429.13	664
	スレッド 2	538.95	748.4
映像の受信・再生あり	スレッド 0	2182.58	20750.76
	スレッド 1	2265.87	18254.28
	スレッド 2	2734.17	21035.91

以上の結果から，本研究の対象システムでは，動画の受信から再生に至るまでの処理がノード 2 上での CPU コアの割り当てに影響を及ぼし，その割り当てによる処理が車両ステータスの通信に遅延をもたらしていると考えた．

3. CPU コア割り当て機能付き RTOS エミュレータ

3.1 CPU コアの割り当て方法

評価実験の結果を踏まえ，本研究では対象システムのノード 2 上で動作する RTOS エミュレータを拡張し，制御系メッセージを処理するプロセスを特定のコアに割り当てる機能を追加した．具体的には，Linux カーネルが提供する sched_setaffinity()システムコール[3]を使用して，RTOS エミュレータから呼び出されるスレッドが同じコアで実行されるような機能を実装した．

図 3 に，機能追加後のノード 2 上で動作するソフトウェアの構成を示す．

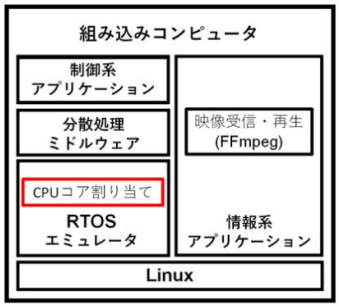

図 3 ノード 2 で動作するソフトウェアの構成

3.2 評価

表 2 に CPU コア割り当て機能を追加した RTOS エミュレータを使用して対象システムを動作させた場合の，車両ステータスのメッセージ往復時間を示す．

表 2　機能追加後のメッセージ往復時間

	Thread	平均値(μsec)	最悪値(μsec)
提案手法	Thread 0	1087.3	8036.85
	Thread 1	1366.3	8719.44
	Thread 2	1193.17	3419.64

表 1 の結果と比較すると，メッセージ往復時間は平均値，最大値ともに大幅に削減できていることがわかる．また，メッセージ往復時間の半分を片道分の時間と考えた場合は，表 2 における片道分の最大値が 4.35[msec]程度となるが，自動車の制御システムが 10~100[msec]の周期で動作するものが多いことや，車載インフォテインメントシステムの用途であることなどを考慮すると，実用上問題ない値であると考えられる．

3.3 CPU 負荷に応じた動的コア割り当て機能

3.1 で述べたコア割り当て機能はシステムの設計者が静的にコア数を決定するものであったが，本研究では CPU の使用率に応じて動的にコア数を増減する機能の追加も行った．具体的には Linux 上でシステムの動作状況が保存されている/proc/stat より一定周期ごとに CPU 使用率を取得し，その値に応じてコア数を増減させる機能を RTOS エミュレータに実装した．その後 3.2 と同様の環境で複数回実験を行ったところ，表 1 のコア割り当てを行わない場合と比べるとメッセージ往復時間は改善する傾向がみられた．

しかし，コアを増減させる際のオーバーヘッドが大きく，静的にスレッドを特定のコアに割り当てる場合よりもメッセージの往復時間が増大したことや，コアの切り替えが発生した場合の動作が非常に不安定となる問題が確認できたことから，動作中のスレッドに動的にコアを割り当てる機能は，本研究が対象とするシステムには適さないと考えられる．

4. おわりに

制御系メッセージ(車両ステータス)と情報系メッセージ(映像データ)の融合した通信に対応したシステムを実現するため，車載インフォテインメントシステムを再現し，実験を行った．実験の結果から CPU コア割り当てによる影響が大きいことに着目し，RTOS エミュレータから呼び出されるスレッドが特定のコアを使用する手法を検討した．

RTOS エミュレータを拡張して提案手法を実装し，実験を行ったところ，車両ステータスの通信の QoS が大幅に改善され，メッセージの往復時間は実用上問題のない範囲であることが確認できた．

しかし，本論文で述べた CPU コアを増減させる方法では，CPU コア間で共有される資源の干渉が考慮できないという課題がある．そこで，マルチコアプロセッサに搭載される，記憶階層を持つキャッシュメモリにおいて，CPU コア間で共有されるキャッシュメモリに着目した研究を行う予定である．

謝辞

本研究で使用した TOPPERS/ATK1 の開発者に感謝する．

参考文献

[1] Onur Alparslan, Shin’ichiArakawa, Masayuki Murata. Next generation intra-vehicle back-bone network architectures. In 2021 IEEE 22nd International Conference on High Per-formance Switching and Routing (HPSR), pp. 1–7, 2021.
[2] OSEK/VDX, “OSEK/VDX Operating System”, Version 2.2.3, 2005.
[3] Man page of SCHED_SETAFFINITY, https://linuxjm.sourceforge.io/html/LDP_man-pages/man2/sched_setaffinity.2.html