システムの記事

車載システムにおける高信頼性と電力管理機能の実現

――ソフトウェアにもフェイル・セーフの考えかたが必須

岡澤幸一

ここでは車載機器の，特にソフトウェアの面から見た信頼性の向上と電力管理の技術について述べる．ソフトウェアの場合，バグをゼロにすることは不可能に近い．それでもシステムが正常に動作し続けるように設計しなければならない．また，自動車に搭載される電子機器が増えるにしたがって，消費電力についても十分考慮して設計する必要がある．　（編集部）

民生用の組み込み機器として車載装置が注目されています．現在のところ，物理的なネットワーク・レイヤを何にするか，情報プロトコルをどうするかといった議論が盛んに行われているようです．

これらの議論の後，実際に車載装置を設計する段階になると，次は何が検討事項になるのでしょうか．それは，「車載」という条件のもと，ハードウェア（ROM）に組み込まれるソフトウェアに要求される機能だと思います．そこでは，次の2点が重要なポイントとなります．

- 高信頼性のための機能
- 電力管理のための機能

ここではこの2点について，システム側からの要求事項や実際のOSの実装という視点で考えてみたいと思います．

1 ソフトにおけるフェイル・セーフ機能とは

車載システムでは高信頼性が要求されます．これはあたりまえの話といえますが，では実際にどのようなことが求められるのでしょうか．ひと言で言うと，「システムの機能が『停止』しない」という点に集約されます．

ソフトウェア・システムの場合，完全にバグを除去することはかなり難しいことです．しかし，かりにバグが原因のトラブルが発生したとしても，システムとしての機能は継続しなければなりません．自動車に限らず，機械分野においてフェイル・セーフという機能は重要です．このことは，ソフトウェアにも求められます．

●ソフトウェアの信頼性を示す三つの指標

高信頼性を実現するための基本的な項目は次の三つです．

1）堅ろうさ

同じレベルの障害が発生したとき，システムがダウン（停止）する場合としない場合があります．このとき，停止しないシステムのほうが「より頑健である」，あるいは「堅ろうさを持っている」と言います．

ここで，汎用OSの基本的な障害であるメモリの不正アクセスを考えてみましょう．メモリ障害が検知されたとき，汎用OSでは不正アクセスを行ったプロセスがセグメント・フォールト[注1]で停止します．この場合，被害はユーザ・プロセスにのみ限定することができ，ほかのプロセスは走行しています．つまり，メモリ障害の検知機能を持ったOSは堅ろうさを持っていると言えます．

2）異常の検知

上記のメモリの不正アクセスの例では，メモリ障害を検知する機能が重要な役割を果たしています．ここでは，アルゴリズム的なバグを考えてみましょう．あるコード内でループとなり，次のステップに移らないようなケースがあります．この場合，プログラムとしては動作を継続していますが，動作そのものは異常になっています．

物理的な異常検知はOSの機能で実現することが可能で

注1：CPUが検知する基本的な障害によってアクセスできない領域をプログラム・コードがアクセスした場合，通常，CPUはセグメント・フォールト用のトラップ・ベクタにジャンプする．それ以降はOSが受け取り，セグメント・フォールト障害として対応する処理（シグナル通知）を行う．

すが，システム的な異常（例えば，ハードウェアにアクセスする順番やタイミングの異常）を検知する場合，プログラムの中になんらかの機能を組み込む必要があります．システムを構築するたびに検知機能を組み込むのではなく，フレームワーク[注2]として提供された環境を使うことができれば，ソフトウェア開発者の手間を軽減できます．

3）異常からの復旧

異常が検知され，その異常が発生したプログラムを停止させた後は，復旧作業が必要となります．復旧手段のいちばんシンプルな例として，ハードウェアのウォッチドッグ・タイマを考えてみます．これは，ソフトウェアからの一定時間の反応がなくなると，強制的にリセットをかける回路をシステムに組み込むことで実現されます．この方法は，一見，乱暴にみえますが，システムを「停止させない」という点では，最後の「とりで」としてもっとも信頼できる手段となるわけです．

ウォッチドッグ・タイマの機能は，次の二つに分けることができます（**図1**）．

- 異常の検知：ソフトウェアからの反応が一定時間ない
- 異常からの復旧：システムに対する物理的リセットの発生

実際には，**図1（b）**のようにソフトウェアからの周期的なアクセスとCPUのリセット端子をつなぐ簡単な回路で実現できます．このため，さまざまなシステムに搭載されています（かりに搭載されていないとすれば，信頼性の最後のとりでを持っていないとも言える）．

ここで問題となるのは，リセットによる異常復旧にかかる時間です．ポート・アクセスのみのシンプルな組み込みシステムなら，リセット，リブート，システムの再スタートにかかる時間はさほど大きくありません．しかし，現在の組み込みシステムでは各種デバイス，ネットワーク，マルチプロセスの環境が複雑に絡み合っており，システムが要求する以上の復旧時間がかかります．

ここで，構造（カーネル）の異なる以下の二つのOSについて，復旧時間の違いを考えてみます．

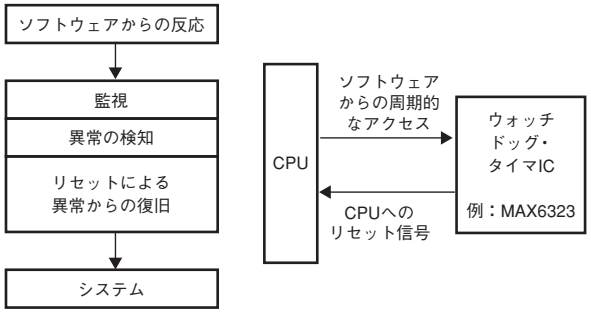
- モノリシック・カーネルOS
- マイクロカーネルOS

モノリシック・カーネルでは，簡単に言うとドライバとシステム・サービスがカーネルに組み込まれています．この構造では，アプリケーションに異常が発生した場合はアプリケーションを停止して再起動すればよいのですが，システム・サービスやドライバの異常については，カーネル自体が停止してOSをリブートしなければ復旧できません（**表1（a）**）．UNIX，組み込みLinuxなどの汎用OSがこのタイプとなります．

これに対して，マイクロカーネルでは，ドライバとシステム・サービスはカーネルの外に通常のアプリケーションと同じレベルで存在します．この構造をとると，アプリケーション，システム・サービス，ドライバのどれが異常停止しても，問題となる箇所だけを再起動すれば復旧できます（**表1（b）**）．ただし，もっとも深いレベルで問題が発生したときは，OSのリブートが必要になります．筆者ら（カナダQNX Software Systems社）が開発したリアルタイムOS「QNX」がこのタイプです[注3]．

●OSの起動に時間がかかりすぎる

車載システムにおいて，反応速度，発熱量，電力消費量



（a）ウォッチドッグ・タイマの機能　　（b）実際の構成

〔**図1**〕ウォッチドッグ・タイマ

ウォッチドッグ・タイマは，ソフトウェアから一定時間の反応がなくなると，システムに対して強制的にリセットをかける．

〔**表1**〕OS構造による復旧作業の違い

（a）モノリシック・カーネルの場合

異常発生箇所	現　象	対　応
アプリケーション	アプリケーションの停止	アプリケーションの再起動
システム・サービス	カーネルの停止	OSのリブート
ドライバ	カーネルの停止	OSのリブート

（b）マイクロカーネルの場合

異常発生箇所	現　象	対　応
アプリケーション	アプリケーションの停止	アプリケーションの再起動
システム・サービス	サービスの停止	サービスの再起動
ドライバ	ドライバの停止	ドライバの再起動

注2：ソフトウェア・システムを設計，実装する際の「枠組み」を指す．実装におけるソフトウェア部品（ライブラリ，マネージャ）も含む．
注3：マイクロカーネルのOSとしては，QNXのほかに，「BeOS」や「Chorus」，「Mach」，「NeXT」，「Mac OS X」などがある．