第3章　プロセッサ間の同期をソフトウェアで実現する方法

マルチプロセッサでプログラムを作成するためのアセンブリ命令

中森 章

マルチプロセッサ・システムを構築する際には，資源の排他制御をどのような手法で実現するのかが問題となる．近代的なプロセッサでは，排他制御を1命令で実現できるアセンブリ命令が用意されている．本章では，これらのアセンブリ命令について解説する．

本章では，マルチプロセッサ間で同期を取るために用意されたアセンブリ言語の命令について解説します．また，命令だけにとどまらず，その使い方を知ることにより，資源の排他制御を実現する方法などについても解説します．

マルチプロセッサ・システムにおける排他制御

マルチプロセッサ・システムを構築する上では，いかにして排他制御を行うかが重要になります．純粋にソフトウェアだけで排他制御を行うことも可能ですが，ハードウェアによる支援があると，高速な排他制御（1クロック～数クロックで実行）が可能です．そして，現在のマルチプロセッサ，マルチコアCPUのほとんどがハードウェア的な排他制御機構を持ち，そのためのアセンブリ命令を備えています．

● 同期のためのソフトウェア処理

マルチプロセッサ・システムにおいては，共有バスがメイン・メモリ（共有メモリ）にアクセスする唯一のアクセス手段です．メモリの排他制御は，バスの使用権を獲得したプロセッサがメモリ・アクセスを独占し，ほかを閉め出せばよいのです．この処理は，不可分（アトミック）なスワップ命令（データ交換命令）があれば簡単に実現できます．

不可分とは，あるプロセスがメモリにアクセスしている間に，ほかのプロセッサからのメモリ・アクセスがないことをいいます．あるいは，ほかのプロセスがアクセスしようとしてもそれが禁止されている状態をいいます．実際には，（バス）ロックという端子を活性化して外部回路に通知し，ハードウェア的にほかのプロセッサのアクセスを禁止します．

話が横にそれましたが，ソフトウェア的には，排他制御のために，各共有資源に対して，1対1に対応するロック変数を用意します．つまり，ハード・ディスクには変数A，ディスプレイには変数B，プリンタには変数Cといった具合です．ここで，ロック変数が0の場合はそれに対応する資源が未使用，0でない場合はそれに対する資源が使用中であることを示すものとします．

プロセスは早い者勝ちで，0以外の値をロック変数に書き込むことで，その資源を使用する権利を得ます．例えば，プリンタを使用したい場合，変数Cが0なら誰も使用していないので使用可能，0以外なら使用中なので待つ（またはあきらめる）といった処理になります．

初期のLinuxでは，共有資源ごとにロック変数を持つのではなく，ただ一つのロック変数で排他制御を実現していたそうです．つまり，ロック変数にアクセスできたプロセスは，すべての共有資源にアクセスできるようになるのです．こういう単純な制御でも，システム的には破たんしない程度の実行性能が得られる（昔は得られていた？）のでしょうね．

● テスト・アンド・セットによる排他制御

排他制御を実現する最も単純な処理が，「テスト・アンド・セット」です．これは，ロック変数を読み込み，その値が0ならそこに1（または0以外の値）を書き込みます．ロック変数が0でなければ，書き込みは行いません．通常，ロック変数の前の値がリードの結果