6P-3

多重ＯＳの論理資源多重方式の設計

竹内　律子、鈴木　茂、岸田　一
沖電気工業株式会社

1　はじめに

我々が設計している多重OSシステムでは、各OSのモデルとセマンティクスの独自性が高く、バーチャルマシンによってハードウェア資源を共有し、いくつかのソフトウェア資源を抽象データ型で定義して共有する。各抽象データ型は、資源アクセス管理と資源待ちキュー管理からなり、それぞれ資源アクセスメカニズムと待ちキュー管理ポリシを提供する。なお、本論文では、抽象データ型を実現する論理資源管理モジュールについてのみ述べ、他のソフトウェアモジュールおよびバーチャルマシンの実現方法については述べない。

2　背景

多重OSシステムは、既にいくつか開発され、稼働している。　それらは、CPU，メモリ、I/Oデバイス、割り込みなどのハードウェア資源をOS間で共有するためにバーチャルマシンを提供し、その上で複数のOSをコンカレントに走らせることができる。ソフトウェア構成的には、あるOSが他のOSをエミュレートする、または各OSは各々バーチャルマシンを割り当てられて独立している。我々は、ハードウェア資源を仮想化させ、その上でモデルの非常に異なるOSを多重化させる。さらに、ファイル、ディレクトリ、バッファ、プロセス間通信メッセージ、セマフォなどのいくつかの資源（以降、論理資源と呼ぶ）を共有する。モデルの非常に異なるOSを多重化させて論理資源を共有するとき、その問題点を以下に示す。

あるプロセスが論理資源を取得しようとしたとき、その論理資源が利用可能でないならば、そのプロセスはそれが利用可能になるまで待たされる。論理資源管理モジュールは、資源待ちキューにそのプロセスをつなぎ、そのプロセスの属するOSのプロセス管理にそのプロセスのスリープ要求を出す。結果として、一つの論理資源待ちキューには複数のOSタイプのプロセスがつながることになる。論理資源管理モジュールから見れば、各OSによってスリープ、ウェークアップ、シグナルの送受信、プロセス状態遷移のメカニズムとセマンティクスが異なる。各OSのプロセス管理から見れば、資源待ちプロセスキューが個々の論理資源によって管理され、各OSの管理外になる。以上述べたことは、単一OSシステムや統一的なプロセスモデルがある場合には起こらず、複雑なOSを独立性高く多重化し、論理資源を共有するときに起きる。

3　論理資源管理方式

（１）システム概要

システム全体の概要を図１に示す。各モジュールの主な機能を以下に述べる。CPU管理モジュールはCPU割り当ての管理をする。メモリ管理モジュールは実記憶および仮想記憶の管理をする。I/O管理モジュールはブロックデバイス、キャラクタデバイスなどのデバイスドライバを提供する。割り込み管理モジュールはハードウェア割り込みの管理をする。上記モジュールは、多重化されているOS間でハードウェア資源を共有できるように、各OSにバーチャルマシンを提供している。このバーチャルマシン上に、各OS固有のプロセス管理モジュールとファイル管理モジュール、および各OS共通の論理資源管理モジュールがある。プロセス管理モジュールはプロセス状態管理、プロセススケジューリングをする。ファイル管理モジュールはファイルアクセス機能を提供する。

（２）論理資源管理モジュール

論理資源管理モジュールは、メッセージバッファ、ファイルキャシュバッファ、ファイル、ディレクトリ、セマフォなどの抽象データ型をインプリメントする。各論理資源管理モジュールは図２に示すように２つの部分、論理資源アクセス管理モジュールと論理資源待ちキュー管理モジュールからなる（以後、それぞれＬＲＡＭ、ＬＲＱＭと呼ぶ）。ＬＲＡＭは、資源アクセスのためのインタフェースをプロセスに提供し、必要に応じて排他制御をしながら資源の取得と解放を行う。ＬＲＱＭは、資源待ちキューアクセスのためのインターフェースをＬＲＡＭと各OSのプロセス管理に提供し、プロセスをどういう順序でキューにつなぎ、どううういう順序でキューから外すかに責任を持つ。ＬＲＡＭは資源割り当て管理を抽象データ化したものであり、ＬＲＱＭは資源待ちキューを抽象データ化したのもである。

論理資源管理モジュールをＬＲＡＭとＬＲＱＭに分離した理由は、資源を取得／解放する事とだれに空いている資源を与えるかを決定する事は別問題であり、かつ資源待ちキューを抽象データ化することによって各OSのプロセス管理は待ちキュー中の他のOSの存在を意識せずに間接的に待ちキュー管理できるからである。前者はポリシ／メカニズムのセパレーションと呼ばれているものである[1]。結果として、ＬＲＱＭはキュー管理ポリシを柔軟に入れ替えたり、複数のＬＲＡＭでポリシの共有ができる。また、プロセス管理は強制的なウェークアップが生起したとき、ＬＲＱＭに資源待ち解除を要求することによって、そのプロセス管理独自のセマンティクスを維持しながら資源待ちを解除できる。

4　処理シーケンス例

共有される論理資源は、ＬＲＡＭが提供するアクセスインタフェースを持っている。代表的な例として、セマフォではP,V,CP、バッファキャシュではgetblk，readblk，writeblk、などである。一方、ＬＲＱＭは以下のインタフェースを持っている。

System Design for Sharing Logical Resources in a Multi-Operating System

R. Takeuchi, S. Suzuki, H. Kishida
Oki electric industry, Co., Ltd.

・ＰＵＴ：　指定されたプロセスを資源待ちキューにつなげる
・ＣＨＯＯＳＥ：　ＬＲＱＭのポリシにしたがって任意の１つの待ちプロセスを選ぶ
・ＲＥＭＯＶＥ：　指定された待ちプロセスを資源待ちキューからはずす

ＬＲＡＭ、ＬＲＱＭ、プロセス管理の関係を図３に示す。あるプロセスがＬＲＡＭに資源を要求するとき、資源が利用可能ならばそのプロセスに資源を与える。資源がない場合、ＬＲＡＭはそのプロセスのプロセス管理のsleepによってそのプロセスをスリープさせる。同時に、ＬＲＡＭはPUT(pid)によってそのプロセスを資源待ちキューにつなぐ。どこにそのプロセスをつなぐかはＬＲＱＭのポリシである。ＬＲＡＭはあるプロセスが資源を解放したとき、CHOOSE()によって資源待ちプロセスを取り出してもらう。どのプロセスを取り出すかはＬＲＱＭのポリシである。同時に、ＬＲＡＭは取り出されたプロセスのプロセス管理のwakeupによってそのプロセスをウェークアップする。ＬＲＡＭがキャンセル機能を持っていて資源待ちがキャンセルされた場合、REMOVE(pid)によってそのプロセスを資源待ちキューからはずす。同様に、プロセス管理が、プロセス管理の理由で資源待ちしているプロセスをウェークアップする場合、REMOVE(pid)によってそのプロセスを資源待ちキューからはずす。

CHOOSEには、キュー管理ポリシが反映される。代表的なポリシはFIFOであるが、プライオリティによる制御、待ちプロセスをすべて起こす場合、可変長ブロックの割り当てでブロックサイズの合うプロセスを選ぶ場合などにも柔軟に対応できる。なお、ＬＲＱＭはＬＲＡＭの内部データを参照してCHOOSE時の意思決定につかてもよい（インプリメンテーションの完全な分離を意味しない）。異なる資源を待つプロセスが同じキューにつながれても問題がなければ、ＬＲＡＭとＬＲＱＭは必ずしも１対１に対応してなくてもよい。図４にセマフォのインプリメント例を示す。

５　まとめ

論理資源管理モジュールのコーディングを終了し、この部分を含めシステム全体のデバッグにはいる。

参考文献

[1]Levin, R., Cohen, etc: "Policy/Mechanism Separetion in HYDRA", Proc. 5th symp. on Operating System Principle, 1975, pp. 132-140

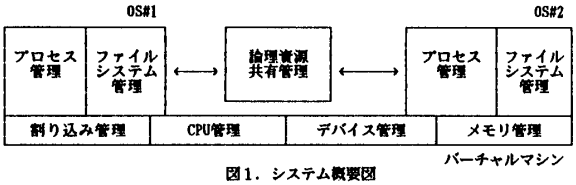


図1．システム概要図

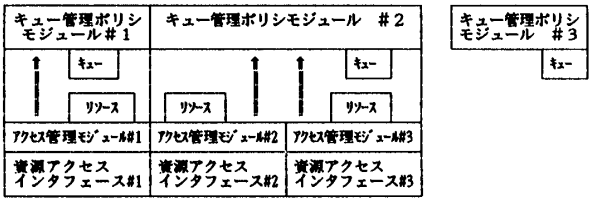


図2．論理資源管理モジュール

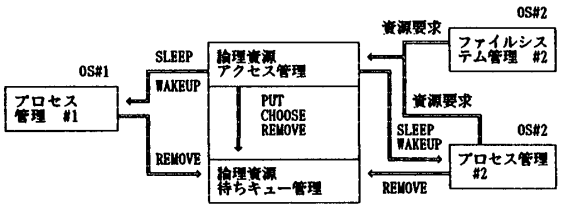
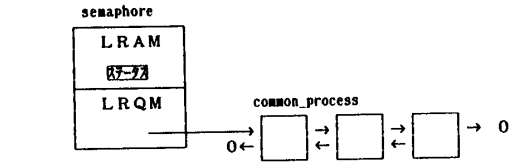


図3．モジュール間インタラクション




```
LRAM
P() {
        LRAM.LOCK();
        if(ステータス  ==  ロック中)
                LRQM.PUT();
                LRAM.UNLOCK();
                COMMON_PROCESS.SLEEP();
        else {
                ステータス  =  ロック中
                LRAM.UNLOCK();
        }
}
V() {
        LRAM.LOCK();
        if(LRQM.CHOOSE() == 該当プロセスなし)
                ステータス  ==  空き
        else
                COMMON_PROCESS.WAKEUP();
        LRAM.UNLOCK();
}
LRQM
REMOVE(common_process* p) {
        LRQM.LOCK();
        キューからはずす.
        LRQM.UNLOCK();
}
PUT(){
        LRQM.LOCK();
        キューの最後にプロセスをつなぐ.
        LRQM.UNLOCK();
}
CHOOSE() {
        LRQM.LOCK();
        if(キューにプロセスがある){
                先頭のプロセスをキューからはずす. /* FIFOポリシ */
                LRQM.UNLOCK();
                return(0);
        }
        LRQM.UNLOCK();
        return(1);
}
```

図4．セマフォの実現例