並列論理型ＯＳ－ＰＩＭＯＳ（１）

― 資源管理方式 ―

4D-3

佐藤裕幸，越村三幸，近山隆，滝和男
（財）新世代コンピュータ技術開発機構

１．はじめに

第５世代コンピュータの研究開発プロジェクトでは，並列推論マシン用のオペレーティング・システムPIMOS(Parallel Inference Machine Operating System)[1]をマルチPSI第２版[2]上に開発中である．PIMOS は，ユーザが消費するCPU 時間やメモリ等の資源を管理し，入出力装置等をユーザに提供する．また，ユーザが過ちを犯してもシステムがダウンしないように監視する．しかし，このシステムの保護を並列環境の下で行なうのは，容易ではない．

本稿では，入出力装置を含めたさまざまな資源を管理する上で，ユーザの誤りからシステムをいかに保護するかについて報告する．

２．「荘園」の機能

ユーザが入出力装置へアクセスする等といったことを行なうには，PIMOS に要求を出さなければならない．このようなユーザとPIMOS との通信は，共有変数の具体化によって行われる．従って，ブート・プログラムは，以下のような形になる．

```
boot :- true | pimos(Req), user(Req).
```

ユーザとPIMOS は，共有変数Req をストリームとして利用することで，通信を行なう．PIMOS の記述言語であるKL1(核言語第１版)は，Flat GHCを基本にしており，すべてのゴールがAND 関係となっている．従って，ユーザ・プログラムのどれか１つのゴールが失敗してしまうと全体(boot)の失敗となり，システム・ダウンとなってしまう．またこれでは，ユーザ・プログラムが暴走した時に強制終了させることや，実行時間等を計測することができない．そこでPIMOS の記述言語であるKL1 では，このようなユーザ・プログラムの実行過程をメタなレベルから監視する機能を持った組込述語を用意している．

```
execute(GOAL, CONTROL, RESULT)
```

ここで，各引数は以下のようなものである．

(1) ＧＯＡＬ

実行されるゴール列であり，コンパイル時に呼び出し先が決まっていなければならない．execute では，このゴールの実行中に消費される資源を管理し，実行中の異常に対しては異常処理の機能が働く．この実行に関わる資源管理／異常処理の単位を「荘園」と呼ぶ．荘園内で再びexecute を実行してもよく，荘園は任意にネストできる．

(2) ＣＯＮＴＲＯＬ

資源割当制御ストリームであり，各種のコマンドを送ることにより，以下のような資源の制御が行なえる．

・荘園の実行の開始／中断／放棄
・荘園の実行中に消費してもよい資源量（CPU 時間やメモリ量等）の指定
・荘園の実行中に消費した資源量の問い合わせ

資源の割り当てはいずれも，ネスティングの外側の荘園に対する指定が，内側に対しても有効である．例えば，外側の荘園の実行の中断を指定すると，その内側の荘園の実行もすべて中断される．また，ある荘園の資源の許容消費量は，その内部のネストした荘園の消費資源を合計したものに対する制限である．

(3) ＲＥＳＵＬＴ

荘園の実行状態に関する報告が行われるストリームであり，以下のような報告が送られてくる．

・荘園内の実行の成功／放棄
・資源の超過使用
・例外事象

例外事象には，算術演算時のオーバフロー等の他に，ゴールの失敗も含む．例外事象の報告には，以下のような情報が送られてくる．

・どのような例外事象が生じたかについての情報
・例外事象の原因となったゴール
・例外を起こしたゴールの代りに実行するゴールを指定するための変数

また，ソフトウェアによって検出される異常事態を同じ枠組みで処理できるようにするために，ソフトウェアで積極的に例外を起こす機能が用意されている．

この荘園の機能によりブート・プログラムは，以下のように記述できる．

```
boot :- true | pimos(Req, C, R),
               execute(user(Req), C, R).
```

ユーザは，通信用ストリームReq を通じてPIMOS と通信し，PIMOS は制御用ストリームC と報告用ストリームR を通じてユーザ・プログラムの実行を制御する．

３．PIMOSとの通信とシステムの保護

ユーザが入出力を行ないたい場合は，Req に要求を流せばよい．例えば，指定した文字数分キーボードから読み込むには，以下のように記述する．

```
user(Req) :- true |
    Req = [getb(N,String)|ReqT].          ― (a)
```

要求getb/2は，第１引数に読み込みたい文字数を指定すると，第２引数に読み込まれた文字列が返される．従って，N, String もPIMOS との通信用の変数となる．また，PIMOS 側では，以下のように記述する．

Resource Management on PIMOS

H.SATO, M.KOSHIMURA, T.CHIKAYAMA, K.TAKI
Institute for New Generation Computer Technology

```
pimos([getb(N,String)|ReqT]) :- true |
        read_from_kbd(N, KBDString),          -- (b)
        String = KBDString,                   -- (c)
```

PIMOS は要求getb/2を受け取ると、read_from_kbd/2 によりキーボードから読み込み、読み込んだものとユーザに返す変数をユニファイする。従って、変数Stringには文字列がユニファイされることになる。

しかし、このままでは以下のような問題点がある。

・問題点-1

ユーザが以下のように（誤って）文字列が入るべき通信用変数にアトムをユニファイしてしまったとする。

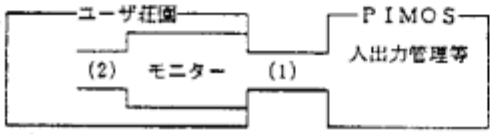
```
String = aaaaa                                -- (d)
```

PIMOS が扱う計算機は並列計算機なので、上記の(b)から(d)までの各ゴールの実行順序は、規定されていない。従って、(d)-(b)-(c) の順に実行された場合は、PIMOS側のゴール(c) が、アトムと文字列とのユニフィケーションとなり失敗する。つまり全体が失敗し、システム・ダウンを起こしてしまう。

・問題点-2

ゴール(a) を呼び出した時点では読み込む文字数N が未定義変数であったとする。ところが、それをユニファイする前にユーザ荘園が死んでしまうと、永久にN の値を待つゴールがPIMOS の中に存在してしまう。

そこでこれらの問題点を解決するために、ユーザとPIMOSとの間に以下のような形でモニターを設けることにした。



ユーザは、PIMOS とつながっている通信用ストリーム(1)に直接要求を送らず、間にモニターを入れてストリーム(2) に要求を送る。このモニターでは、ユーザの要求を絶対に失敗しないような形に変換し、また、PIMOS 側で待ち続けないことが保証されるまで待ってから、(1) に流す。このようにPIMOS 側のストリームを直接ユーザに見せないので、ユーザの誤りがモニターで吸収されるだけでなく、ユーザの悪意を持ったシステム(PIMOS) への侵入さえもモニターが監視して防ぐことができる。具体的なモニターの仕事は、以下のようなことである。

・ユーザ側で値を指定すべきデータは、そこに実際に値が入るまで待ってからPIMOS に要求を送る（問題点-2の解決）。

・PIMOS 側で値を返すデータの部分にはユーザが指定した変数ではなく、必ず未定義であるような別の変数を送り、PIMOS から値が返された後にユーザが指定した変数とPIMOS に送った変数をユニファイする(問題点-1の解決)。

例えば、上記の例のgetb/2用のモニター・プログラムは、以下のように定義できる。

```
monitor([getb(N,String)|ReqT], OSReq) :- wait(N) |
        OSReq = [getb(N,OSString)|OSReqT],
        wait_and_unify(OSString, String),
        monitor(ReqT, OSReqT).
wait_and_unify(PIMOSV, USERV) :- wait(PIMOSV) |
        PIMOSV = USERV.                       -- (e)
```

読み込みたい文字数N （ユーザが値を指定するデータ）に値が入るまで待ってからPIMOS に要求を送る。一方、読み込まれた文字列が入る変数String (PIMOS が値を返すデータ）の部分には、未定義変数(OSString)を送り、PIMOS が値を返した後にユーザが指定した変数(String)とユニファイする。従って、上記(d) のようなユーザの誤りは、ゴール(e) が失敗し、つまりモニターが失敗することになる。このモニターの実行は、ユーザ荘園内で実行されるので、システムがダウンすることはない。このモニターを考慮したブート・プログラムは以下のようになる。

```
boot :- pimos(OSReq, C, R),
        execute(
            ( user(USERReq),
              monitor(USERReq, OSReq) ), C, R).
```

ここで、通信用ストリームだけは、ユーザ荘園が死んだ時に自動的に閉じられるバルブ機構を持たせるよう特別な扱いとした。つまりこの通信用ストリームは、入出力装置という資源であり、メモリ等と同様にユーザ荘園が死んだ時には、自動的に解放されるものとして扱われる。これにより、通信用ストリームを閉じる前にユーザ荘園が死んでも、そのストリームは自動的に閉じられるので、PIMOS 側には、そのストリームから流れてくる要求を待ち続けるゴールはできない。

4. プロトコル定義言語

このモニター・プログラムは、システム(PIMOS) で提供するのであるが、ユーザ／PIMOS 間の通信プロトコルが分っていれば、機械的に生成することができる。そこで我々は、この通信プロトコルを定義する言語を設け、それからモニター・プログラムへ変換するトランスレータを作成することにした。このプロトコル定義用言語では、ユーザ／PIMOS 間の通信で、どのような形のデータがどちらの方向に流れるのか（どちらが具体化するのか）を定義する。例えば、キーボードに関係する通信プロトコルは、以下のように定義される。

```
kbd_manager = stream(kbd_command).
kbd_command = {
        getb(length, buffer) ;
        getl(line) ;
        getc(character) }.
length  = integer.    buffer    = string.
line    = string.     character = integer.
integer = atomic.     string    = atomic.
```

kbd_manager は、kbd_command で定義されるデータが、ユーザからPIMOS に向けて流れるストリームである。

kbd_command には、３種類のものがあり、例えば、getbの第１引数はlengthで定義されるデータ（最終的には、アトミックなデータ）にユーザが具体化し、第２引数はbufferで定義されるデータ（これも最終的には、アトミックなデータ）にPIMOS が具体化する。

5. おわりに

PIMOS では、KL1 に荘園機能を導入し、それを使用することにより、ユーザ・プログラムの実行を制御する。また、ユーザとPIMOS の間にモニターを設けることにより、ユーザのバグによる誤りだけではなく、悪意を持った侵入からもシステムを保護することができる。

＜参考文献＞

[1] 佐藤他：PIMOSの概要－並列推論マシン用オペレーティング・システムの構築－
第３４回情処全国大会,2P-8,1987-3

[2] 渥他　：Multi-PSIシステムの概要
第３２回情処全国大会,2Q-8,1986-3