Standard ML 上でのマルチスレッド処理の実現*

2J－8

神林 靖†
湘南短期大学

1　はじめに

並行処理モデルを実現する際に、マルチスレッドを用いる方法は広く採用されている。マルチスレッドの実装には、カーネルレベルのパッケージとユーザレベルのパッケージの２とおりの方法が採られているが、一般にユーザレベルのパッケージの方が、高価なシステム呼び出しを必要としないだけ効率がよいとされている。ユーザレベルのパッケージは、効率だけでなく、スケジューリング機構や同期機構についてプログラマが選択できる柔軟性も提供している。

Wand は、並列処理系の機能として必須条件として、1) 排他制御、2) プロセスの保存、3) データの保護の３点を指摘している。彼は、一等市民としての継続を用いることにより、プロセスの保存が容易に実現できることを示した。言語処理系が抽象データ型の機構を持つ限り、スレッド・パッケージ内のデータは保護される。彼は、これらの機能を持つ高級言語として Scheme を採用している。

一等市民としての継続が利用でき、かつ強力な抽象データ型機能を有する言語処理系として、Standard ML of New Jersey(SML/NJ) がある。筆者はこの言語を用いて、継続に基づく単一プロセッサ用マルチスレッド・パッケージを開発した。

2　ユーザレベル・スレッド

```
signature THREAD =
sig
  val fork  : (unit -> unit) -> unit
  val yield : unit -> unit
  val id    : unit -> int
  type mutex
  val  mutex   : unit -> mutex
  val  acquire : mutex -> unit
  val  try_acquire : mutex -> bool
  val  release : mutex -> unit
  val  with_mutex : mutex ->
                        (unit -> 'a) -> 'a
end
```

```
signature QUEUE =
sig
  type 'a queue
  val create : unit -> 'la queue
  val enq    : 'a queue -> 'a -> unit
  val deq    : 'a queue -> 'a option
end
functor UniThread (structure Queue : QUEUE
                   structure Mp : MP) : THREAD =
struct
  val ready_lock = Mp.spin_lock ();
  val ready_queue : (unit cont * int)
              Queue.queue = Queue.create ()
  val current_id = ref 0
  val next_id    = ref 1
  fun reschedule (cont,id) =
          (Mp.lock ready_lock;
           Queue.enq ready_queue (cont,id);
           Mp.unlock ready_lock )
  fun dispatch () =
      let val _ = Mp.lock ready_lock
          val thread = Queue.deq ready_queue
      in  Mp.unlock ready_lock;
          case (thread) of
              NONE => raise Empty
            | SOME (cont,id) =>
              (current_id := id; throw cont ())
      end
  fun fork (child : unit -> unit) =
      callcc (fn parent =>
             (reschedule (parent, !current_id);
              current_id := !next_id;
              child ();
              dispatch () ))
  fun yield () =
      callcc (fn cont =>
             (reschedule (cont, !current_id);
              dispatch () ))
  fun id () = !current_id
```

*An Implementation of Multithreads in Standard ML
†Yasushi Kambayashi, Shonan Junior College

```
  fun try_acquire m =
      let val l = get_lock m
      in  Mp.lock l;
          if ((get_flag m) = OFF)
          then (flag_on m; Mp.unlock l; true)
          else (Mp.unlock l; false)
      end
  fun acquire m = if try_acquire m then ()
          else (yield (); acquire m)
end
```

図1. ユーザレベル・スレッドの仕様と実現

SML/NJ によるユーザレベル・スレッドの仕様と実現を、図1に示す。signature とは、モジュール・インタフェースであり仕様を表す。その実現は structure である。functor とは、signature によりパラメータ化されたモジュールの実現である。この例でいうと、signature が QUEUE と MP である。その signature の structure を引数として受けとることにより、スレッド・パッケージ UniThread は実体化される。QUEUE はプロセスの保存に使用されるわけであるが、このようにパラメータ化することにより、スケジューリング政策を選択することができる。また MP はスピン・ロックを司るモジュールで、単一プロセッサ用スレッド・パッケージでは、拡張性のためだけに備えている。

signature THREAD は多くの関数を輸出してるが、中心となるのは fork と yield である。fork は関数を引数に取り、それのスレッドを生成する。スレッドには一意な識別数が割り付けられ、親プロセスと並行して実行される。yield は自分自身をサスペンドし、リスケジューリングを行なう。スレッドは親プロセスの環境を継承するが、この環境が共有メモリであり、この環境を通じて同期が取られることになる。

スレッド・パッケージの実現で中心となるのは、スレッドのスケジューリングを待ち行列中の継続によって行なっている点である。待ち行列を明示的に表現することによって、柔軟なスレッドの管理が可能になる。

```
abstype tid = TID of bool sref * condition
with
  fun jfork f =
      let val m = mutex ();
          val done = sref (false, m);
          val cond = condition m
      in  fork (fn () => (f ();
                          acquire m;
                          sset done true;
                          release m;
                          broadcast cond));
          TID (done, cond)
      end
  fun join (TID(done, cond)) =
      let val _ = acquire(mutex_of_ref done)
      in  await cond (fn () => sget done);
          release (mutex_of_ref done)
      end
```

図2. スレッド・パッケージを用いた fork-join の例

スレッド・パッケージには、お互いが同期を取る機構も必要である。図2は、スレッド・インタフェースを組み合わせて、単純な fork-join 機構が作成できることを示している。

3 まとめ

SML/NJ 上に単一プロセッサ用のマルチスレッド・パッケージを実現することにより、継続を用いた並行処理モデルが有効であることを示した。本処理系は、SPARC/10 の SUNOS4.1.3 上に実装されている。SML/NJ においては、すべての閉包がスタックでなくヒープに取られるので、継続の生成と実行は関数呼びだしと同じ速度で実行され効率がよい。実際 callcc は、ヒープ領域の確保と初期化を行なっているにすぎない。

次の課題は、並列処理マシンの上にマルチスレッド・パッケージを実装することである。Sparc/20 の Solaris2.3 上に実装する予定である。基本的には、MP モジュールを複数プロセッサ用のものに交換することで対応できるので (実行時システムを別にすれば)、マルチスレッド・パッケージへの変更は大規模とならないと予想している。

参考文献

[1] Mitchell Wand. Continuation-based Multiprocessing, *Proceedings of the 1980 LISP Conference*, pp.19-28, 1980.

[2] J.Gregory Morrisett and Andrew Tolmach. Procs and Locks: A Portable Multiprocessing Platform for Standard ML of New Jersey, *4th ACM Symposium on Principles and Practices of Parallel Programming*, pp.198-207 1993.

[3] Eric C. Cooper and J.Gregory Morrisett. Adding Threads to Standard ML. Technical Report CMU-CS-90-186, School of Computer Science, Carnegie Mellon University, 1990.

[4] 舟津佳永, 関口智紀, 山本強. マルチスレッド処理を実現する Lisp 処理系の開発, 情報処理学会第 *49* 回全国大会講演論文集 *(4)*, pp.147-148, 1994.