システムソフトウェアと
オペレーティング・システム 69－18
（１９９５．６．２３）

投機的処理を支援するオペレーティング・システムにおける世界とプロセスの操作

溝淵 雅也　新城 靖　當眞 聡　根路銘 崇
<{miz, yas, ha, nero}@ocean.ie.u-ryukyu.ac.jp>
喜屋武 盛基　翁長 健治
<kyan@ie.u-ryukyu.ac.jp> <onaga@ie.u-ryukyu.ac.jp>

琉球大学　情報工学科
〒903-01　沖縄県西原町千原1番地
電話：098-895-2221　内線3266
Ｆａｘ：098-895-2688

概要　投機的処理とは、その結果が利用されるかどうか確定される前に実行を開始する処理である。本研究では、プロセス単位の投機的処理を支援するオペレーティング・システムを開発している。我々は、多重プログラミング環境で投機的処理を扱うために世界という概念を導入した。世界とは、プロセスやファイルをいれる箱のことである。本論文では、世界オブジェクトとプロセス・オブジェクトのインタフェースと属性、その実現について述べる。

Operations of Worlds and Processes in an Operating System Supporting Speculative Processing

Masaya Mizobuchi　Yasushi Shinjo　Hajime Toma　Takashi Nerome
<{miz, yas, ha, nero}@ocean.ie.u-ryukyu.ac.jp>

Seiki Kyan　and　Kenji Onaga
<kyan@ie.u-ryukyu.ac.jp>　<onaga@ie.u-ryukyu.ac.jp>

Department of Information Engineering
University of the Ryukyus
Nishihara, Okinawa 903-01, Japan
Phone: +81 98 895 2221 Ext.3266
Fax: +81 98 895 2688

Abstract　Speculative processing is processing that is started eagerly before it is known to be required. We are developing an operating system that supports process-level speculative processing. We use the idea of worlds to deal with the speculative processing. A world is a box that contains some processes and files. This paper describes interfaces, attributes, and implementations of world objects and process objects.

1. はじめに

今日では、ワークステーションがネットワークにより結合された環境が広く普及してきた。また、1000プロセッサ規模の高並列計算機が作られるようになってきた。このような環境では、多くの遊んでいる計算機資源が存在する。この計算機資源を利用できれば計算機の処理効率は格段に向上することが見込まれる。

投機的処理(speculative processing)とは、その結果が利用されるかされないか確定される前に実行を開始する処理である。余剰計算機資源を用いて投機的処理を行うことにより、1つのアプリケーションの処理時間を短縮することが可能になる。

我々は、投機的処理を支援するオペレーティング・システムを研究している[1][5]。この研究で扱う投機的処理は、多重プログラミング環境においてファイルの入出力を行うプロセス単位の粗粒度投機的処理である[2][4]。その目的は、遊んでいる計算機資源を利用することにより処理の並列度を上げることである。従来のソフトウェア・レベルの投機的処理は、単一プログラミング環境において値の書き換えの無い文や節を単位とした細粒度投機的処理である。その目的は、同期のオーバヘッドを減らすことである。

我々は、投機的処理を扱うために**世界**という概念を導入したオペレーティング・システムを開発している。世界とは、プロセスとファイルを入れるための箱のことである。世界を用いることによって従来の投機的処理の機能の無いアプリケーションでも投機的処理を行うことが可能となる。本論文では、世界の考え方とその働き、それを実現するオペレーティング・システムを構成する世界サーバとプロセス・サーバについて述べる。

2. 投機的処理と世界モデル

2.1 世界モデル

投機的処理とは、その結果が利用されるかされないか確定される前に実行を開始する処理である。これに対し、投機的処理ではない処理を**必須の処理**(mandatory processing)という。

投機的処理を実現するためには、次にあげる3つの技術が必要である。

(1)隔離：投機的処理の結果は、利用されることが確定するまで必須の処理に影響を与えないようにしなければならない。

(2)出現：投機的処理の結果は、利用されることが確定すると素早く必須の処理に反映されなければならない。

(3)削除：投機的処理の結果は、利用されないと確定すれば速やかに削除しなければならない。

我々は、ファイルという記憶領域とそれを扱うプロセス・レベルの投機的処理を扱う。投機的処理は、今まで関数型言語や論理型言語において利用されてきた。並列PrologのAND並列やOR並列は、投機の一例である[2][4]。関数型言語や論理型言語には、記憶領域の概念が無いため上の3つの技術の実現は容易である。これに対して、手続き型言語には記憶領域の概念があり、その扱いは複雑になる。

我々は、投機的処理を扱うために世界という概念を導入し、ファイルという記憶領域の取り扱いを単純化する。

世界は、プロセスやファイルを入れるための箱のことである(図1)。プロセスは、同じ世界に属するプロセスと通信を行ったり、同じ世界に属するファイルを自由に読み書きすることができる。ある世界に属するプロセスは、異なる世界のプロセスとの通信を行うことはできない。

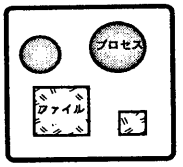

図1.世界の概念図

プロセスは、指定した世界を引き継いだ新しい世界を生成することができる。元の世界を**親世界**、新しく生成された世界を**子世界**と呼ぶ。親世界から子世界の内容を参照することは不可能だが、子世界から親世界の内容を参照することはできる。親世界の内容を変更すると子世界にも影響を及ぼすが、子世界の内容が変更されても親世界の内容は変化しない。また、子世界を生成するとき、複数の親世界を指定することができる。その結果として、世界の親子関係は、DAG(Directed Acyclic

Graph) で表される（図２）。

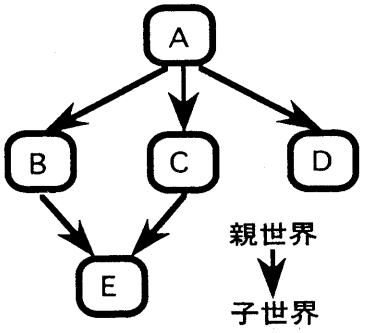

図２．世界のDAG

世界の操作には、生成の他に融合と削除がある。世界の融合とは、指定した２つの世界を１つの世界にすることである。子世界のファイルとプロセスが、親世界に移される。子世界の子は、そのまま親世界の子となる。世界の削除とは、指定した世界を中のファイルとプロセスも含めて削除することである。

投機的処理は、この世界の概念を用いると次のように扱うことができる。ここで、我々は、投機を行う処理を、ファイルから入力し結果をファイルに出力するプログラムに限定する。よって、利用者との対話によって進める処理は、投機的処理として扱わない。本システムでは、投機的処理は管理プロセスによって進められる。

管理プロセスは、子世界を生成する。この作られた世界を**投機の世界**と呼ぶ。これに対して元の世界を**必須の世界**とする。

管理プロセスは､投機的処理を行うプロセス(以下、投機プロセス)を投機の世界に生成する。投機的処理を行った結果、作成・変更されたファイルは、投機の世界に存在する。投機的処理の結果が利用されることが確定した場合、管理プロセスは、必須の世界に投機の世界を融合する。すると、投機的処理の結果であるプロセスやファイルが必須の世界の利用者やプロセスから見えるようになる。一方、投機的処理の結果が利用されないことが確定した場合、管理プロセスは、投機の世界を削除する。すると、投機プロセスや結果のファイルが削除される。

2.2 投機的make

我々は、投機的処理を行う機能を組み込んだmake(投機的make)を開発中である[1][5]。投機的makeは、利用者が"make"と打つ前に記述されたルールにしたがってファイルの最終更新時刻の矛盾を探索する。投機的makeは、ファイルの最終更新時刻の矛盾を検出すると、世界を生成しその中でルールに記述されたコマンドを実行する。投機的makeは、利用者が"make"と打つとその瞬間に生成した世界を利用者の見える世界に融合する。利用者には、瞬時に途中経過や結果が返り、短い時間で処理が行われているかのように見せかけることができる。

3. 世界の概念を導入したオペレーティング・システム

3.1 構成要素

我々の開発している世界の概念を導入したオペレーティング・システムは、次のようなサーバから構成される。

1. 世界サーバ：世界の属性や世界間の関係を管理する
2. プロセス・サーバ：世界の中のプロセスを管理する
3. ファイル・サーバ：世界の中のファイルを管理する

ユーザ・プロセスが発行したシステム・コールは、プロセス・サーバが受け付ける。プロセス・サーバは、受け付けたシステム・コールを解析し、世界サーバやファイル・サーバ、あるいは、自分自身に処理を振り分ける。例えば、世界の中のプロセスがファイルを開くする場合を考えると次のようになる。

プロセス・サーバは、利用者プロセスからファイルを開くシステム・コールを受け付けるとファイル・サーバに開いたファイルを表す開ファイル・オブジェクトを生成するように要求する。ファイル・サーバは、開ファイル・オブジェクトのポインタを返す。この時、ファイル・サーバは、世界サーバに祖先の世界のリストを要求し、そのリストを元に開くべきファイルを探す。ファイルが無ければ、さらに親の世界にさかのぼる。

我々は、世界の概念を導入したオペレーティン

グ・システムを、4.4BSDをもとに開発している。我々が4.4BSDを選んだ理由は、そのカーネルの完全なソースが得られたからである。また、世界を使うことによって、4.4BSD上で動作する投機の機能の無い従来のアプリケーションを投機的処理に利用することが可能になる。

3.2 利用者プロセスによる世界の指定

プロセスが目的の世界を指定するためには、識別子による方法と記述子による方法が考えられる。

識別子による方法では、世界識別子と世界への対応表を世界サーバが保持する。利用者プロセスは、世界識別子を用いて任意の世界を指定することが可能である。UNIXでは、プロセスを指定するためにこの方法を用いている[3]。

記述子による指定では、プロセス・サーバ内に記述子とそれが示す世界の対応表を用意する。UNIXでは、ファイルを指定するためにこの方法を用いている[3]。

我々は、世界を指定する方法として、記述子による指定方法を採用している。その理由は、プロセスが指定できる世界が制限されているため、世界の管理が安全に、かつ容易になるからである。

利用者プロセスは、世界の操作において記述子によって世界を指定する。記述子は、小さい整数である。記述子は、世界が生成されたとき、世界の生成を行ったプロセスごとに割り当てられる。

3.3 世界を操作するシステムコール

世界に関するシステム・コールとして、以下の3つがある。

(1)世界の生成

```
int childwd = world_create
        (int parentswd[], int nwd)
```

指定した複数の世界を親とする新しい世界を生成する。その引数は、世界記述子の配列parentswd[]とその要素数nwdである。世界を生成すると、生成した世界の世界記述子を返す。

(2)世界の融合

```
world_merge(wd1, wd2)
```

指定した2つの世界を融合して1つの世界にする。引数は、融合される世界を示す世界記述子wd1と融合する世界を示す世界記述子wd2の2つである。

(3)世界の削除

```
world_delete(wd)
```

指定した世界を削除する。引数は、削除する世界を示す世界記述子wdである。子孫の世界も同時に削除される。

4. 世界の実現

4.1 世界オブジェクト

世界は、オペレーティング・システムの内部では世界オブジェクトで表される。世界オブジェクトは、カーネルの一構成要素である世界サーバによって管理されている。

世界オブジェクトは、次の手続きを持っている。

- 生成
- 消滅
- 融合
- 比較
- 祖先を求める
- 子孫を求める

生成、消滅、融合の手続きは、3.3節で述べたシステム・コールに対応する。比較は、指定した世界オブジェクトとの間に親子や祖先、子孫の関係があるか調べる。祖先を求める手続きは、親の世界を次々とたどりその一覧リストを返す。子孫を求める手続きは、子の世界を次々とたどりその一覧リストを返す。

祖先の世界の世界オブジェクトを求める手続きは、世界の融合のシステム・コールにおいて用いられる。子孫の世界の世界オブジェクトを求める手続きは、世界の削除のシステム・コールにおいて用いられる。

世界オブジェクトの持つ属性は、世界識別子、所有者の情報（世界を生成したプロセスのプロセス識別子やユーザ識別子）および世界を生成・変更した時刻である。

4.2 世界オブジェクトの実現

世界オブジェクトは、図3に示すデータ構造を

持っている。世界サーバは、世界オブジェクトの各属性を返す手続きを持っている。

世界サーバは、世界オブジェクトを用いて世界の親子関係を管理する。世界の親子関係は、世界オブジェクトが自分の親の世界の世界オブジェクトのリスト(図3のParents)、子の世界の世界オブジェクトのリスト(図3のChildren)を持つことで表される。このように、親子関係のある世界オブジェクト間は、ポインタによって双方向にリンクされている。

世界の祖先を求める手続きは、次のように実現される。

1. 世界オブジェクトの持つ親世界へのポインタを再帰的にたどってすべての祖先を求める。
2. 1.の結果をトポロジカル・ソートする。
3. 重複を取り除く。

子孫を求める手続きは、祖先を求める手続きと同様の手順である。

```
struct world {
  wid_t wid; /* 世界ID */
  pid_t pid; /* 世界を生成したプロセスのID */
  uid_t uid; /* 世界を生成したプロセスの
                                ユーザID */
  gid_t gid; /* 世界を生成したプロセスの
                               グループID */
  struct timeval World_Created_Time;
          /* 世界が生成されたときの時間*/
  struct timeval World_Modified_Time;
        /* 世界の内容が変更されたときの時間 */
  struct List Parents;
          /* 親世界へのポインタ(list )*/
  struct List Children;
          /* 子世界へのポインタ(list )*/
  u_int RefCnt; /* 参照カウンタ */
};
```

図3.世界オブジェクト

5. プロセスの操作

5.1 異なる世界へのプロセスの生成

我々は、異なる世界にプロセスを生成する方法として、次の方法を検討している。

(1)UNIXのforkシステム・コールを拡張し、引数で指定した世界にシステム・コールを発行したプロセスを複製する。さらにexecveシステム・コールにより投機のプログラムを実行する。この方法では、プロセスを別の世界に複製するとき、書き込みモードで開いているファイルをコピーしなければならない。

(2)UNIXのforkシステム・コールとexecveシステム・コールの機能を併せ持つ新しいシステム・コールを作成し、このシステム・コールによってプロセスを生成する。この方法において、親プロセスが子プロセスにファイル記述子を引き渡すために、次の2つの方法を考えた。

(2a)システム・コールの引数により元のプロセスのファイル記述子と新たに作られるプロセスにおけるファイル記述子との対応を指定する方法。

(2b)システム・コールの引数にファイルのパス名を含め、新しく生成したプロセスがファイルのオープンをやり直す方法。

(2b)の方法は、openシステム・コールにおいて名前変換を行うことにより、従来のUNIXのファイル・システム上で世界を扱うファイル・システムが実現ができる。

5.2 世界記述子とプロセス・サーバ

プロセスは、世界を指定するために世界記述子を用いる。プロセスは、オペレーティング・システム内部ではプロセス走行時の情報を格納するプロセス・オブジェクトで表される。その中に世界記述子を管理するworlddesc構造体を導入する。

worlddesc構造体は、図4に示す要素を持つ。

```
struct worlddesc {
    struct world **wd_cworlds;
      /* 生成した世界へのポインタのテーブル */
    struct world *wd_curworld;
      /* プロセスが属する世界 */
    int     wd_nworlds;
      /* 生成した世界の数 */
};
```

図4.worlddesc構造体

プロセス・サーバは、world_create()システム・コールを受け付けると、世界サーバに世界オブジェクトの生成を要求する。世界サーバは、世界オブジェクトを生成し、そのオブジェクトへのポインタをプロセス・サーバに返す。プロセス・サーバは、worlddesc構造体の世界オブジ

ェクトへのポインタの配列*wd_cworlds[]を走査して、空いている要素へポインタを格納する。世界記述子は、この世界オブジェクトへのポインタの配列*wd_cworlds[]の指標である。プロセスは、この世界記述子を用いて新しい世界を指定することができる。

プロセスがどの世界に属しているかの情報は、worlddesc構造体が持っている。この情報は、世界の操作やファイルへのアクセスのときに参照される。

5.3 異なる世界にプロセスを生成するシステム・コール

異なる世界にプロセスを生成する方法は、5.1節で述べた。その方法に基づいて次の3通りのシステム・コールが考えられる。

(1)異なる世界へプロセスを複製する。

```
wfork(wd)
```

指定した世界にシステム・コールを発行したプロセスを複製する。

(2)異なる世界へプロセスを生成する。

a)
```
process_create(wd, path, argv, env,
                pattr, fd[], nfd)
```

指定した世界の中にプロセスを生成する。引数は、世界記述子、生成するプロセスのpath名、生成するプロセスの引数、環境変数、プロセス属性、ファイル記述子の配列、ファイル記述子の配列の要素数である。このファイル記述子fd[]は、現在のプロセスのファイル記述子と新しいプロセスのファイル記述子の組である。

b)
```
process_create(wd, path, argv,
          env, pattr, fdfile[], nfd)
```

指定した世界の中にプロセスを生成する。引数は、世界記述子、生成するプロセスのpath名、生成するプロセスに渡す引数、環境変数、プロセス属性、ファイル記述子の配列、ファイル記述子の配列の要素数である。引数fdfile[]は、新しいプロセスのファイル記述子、ファイル名、ファイルをオープンするときのフラグとモードの構造体である。

6.おわりに

この論文では、投機的処理を扱うため世界の概念を導入したオペレーティング・システムにおける世界とプロセスの操作について述べた。世界は、オペレーティング・システムの内部では、世界オブジェクトで表される。世界関係を示すDAGは、世界オブジェクトを操作して構成される。世界を操作するシステム・コールは、世界オブジェクトの操作によって実現できる。また、違う世界にプロセスを生成する方法について述べた。

今後は、世界を導入したオペレーティング・システムにおけるファイル・サーバと投機的makeの詳細な検討を行い、実現を進めていく。

参考文献

[1]根路銘、當眞、新城、喜屋武、翁長:“粗粒度投機的並列処理を支援するオペレーティング・システムの構想”, SWoPP'94 情報処理学会研究報告, 94-ARC-107, pp.89-96 (1994)

[2]R.B.Osborne: "Speculative Computation in Multilisp", Proc. US/Japan Workshop on Parallel Lisp ("Parallel Lisp: Languages and Systems", Springer-Verlag LNCS 441, pp.103-135, 1990)

[3]Samuel J. Leffler, Marshall Kirk McKusick, Michael J. Karels, and John S.Quarterman: "The Design and Implementation of the 4.3BSD UNIX Operation System", Addison-Wesley Publishing Company, Inc. (1989)

[4]R.H.Halstead,Jr.: "New Ideas in Parallel Lisp: Language Design, Implementation, and Programming tools", Proc. US/Japan Workshop on Parallel Lisp ("Parallel Lisp: Languages and Systems", Springer-Verlag LNCS 441, pp.2-57 (1990))

[5]根路銘、當眞、新城、喜屋武、翁長:“粗粒度投機的並列処理支援オペレーティング・システム上で動作する投機的makeの世界操作”, 1995年 電子情報通信学会総合大会, D-168, p.176 (1995)