Cプログラムのためのプロセス間通信に基づくテスト基準について

5T−5

— 並行処理プログラムにおけるセマフォテスト基準 —

伊東 栄典　古川 善吾　牛島 和夫
九州大学 工学部 情報工学科

1. はじめに

並行処理プログラムが普及するに伴い、プログラムの信頼性向上手法としてのテストが重要となっている。逐次処理プログラムに比べ並行処理プログラムは動作が複雑であるため、新しいテスト法が必要である。現在一般的に使用されているプログラミング言語として、C言語がある。このC言語には元来並行処理を記述する能力は無い。このため、C言語で書かれたプログラムはオペレーティングシステムとの連携により並行処理を実現している。C言語で書かれたプログラムにおいてオペレーティングシステムとの連携は、ライブラリとして用意されているシステムコールの呼び出しによって行なわれる。そのために、オペレーティングシステムと連携したプログラムに対しては、言語だけでなくライブラリに対応したテスト法と支援システムが必要になる。これまで我々の研究室ではAdaを対象に、遠隔手続き呼びだし、共有変数に基づくテスト充分性評価方法の検討を行なってきた[1, 2]。

今回、C言語、UNIXオペレーティングシステム[3]に対するテスト充分性評価の可能性を明らかにするためにSYSTEM V系のオペレーティングシステムで標準的に備わっているセマフォに対するテスト充分性評価法を考察した。今回の発表では、その定義と意味及び計測可能性について述べる。

2. 定義

2.1 セマフォ

セマフォはDijkstraにより導入された機構である[4]。セマフォは実現しやすく十分に強力であるため、並行プログラミングの問題に対してエレガントな解を与えるのに使うことができる。

セマフォは、セマフォ変数semidとそのsemidに対する2つの操作、P命令とV命令からなる。これらの操作の定義は以下の通りである。

[定義 1] P命令、V命令

- P(semid)
  if (semid > 0)
  then semidの値を1つ減らす。
  else プロセスの実行を一時停止。
- V(semid)
  if (待機中のプロセスのキューが空でない)
  then キューの最初のプロセスを再開。
  else semidの値を1つ増やす。

これらの命令はSYSTEM Vでは、システムコール*semop*を用いて実現される[3]。このセマフォに対する操作*semop*ではパラメータによって、P命令とV命令を区別する。

Testing criteria using interprocess communication of C programs
— Semaphore testing criterion —
Eisuke ITOH, Zengo FURUKAWA and Kazuo USHIJIMA
Kyushu University

2.2 被覆率(Coverage)

並行処理プログラムのテスト充分性を評価するために、被覆率C(Coverage)を次のように定義する。

[定義 2] 被覆率C

$$C = \frac{|W|}{|M|}$$

ただし、Wはプログラムを実行したときに発生する事象の集合、Mはプログラムに含まれる測定対象となる事象の集合、$||$は集合の要素数を表す。

被覆率Cを計測するためには、実行開始前にMが定義できなければならない。また、被覆率に基づくテスト基準では、一般に、被覆率を100%にすることを要求する。

3. セマフォテスト基準

被覆率に基づくテスト基準を次に定義する。そのためにプログラムに含まれる測定対象の集合を定義し、それに対する被覆率を100%にすることをテスト基準とする。

2.1節で述べたように、UNIXでは、*semop*システムコールによってセマフォに対する操作が実現されている。故に、まず、Sem_0テスト基準としてセマフォに対する操作を測定対象とする。

[定義 3]　$Sem_0 : M = \{semop\}$

Sem_0テスト基準は、セマフォに対する操作が少なくとも1回は実行されることを必要とする。テスト基準の複雑さを測定対象の数で表すと、Sem_0テスト基準の複雑さは、*semop*の数である。すなわち、*semop*の総数をnとすれば複雑さのオーダーは$O(n)$となる。

セマフォに対する操作*semop*は、セマフォ変数の値、あるいはプロセスキューの長さに応じて、動作が異なっている。そのために、テスト充分性を評価するためには、*semop*の実行だけでなく、*semop*の動作を全て実行する必要がある。そこで、次のようなセマフォ命令semcomを定義し、それを用いてSem_1テスト基準を定義する。

[定義 4] セマフォ命令semcomは次の2つ組である。

$$\text{semcom} = (semop, b)$$

ただし、bは、セマフォ変数semidが0またはプロセスキューが空でない時1であり、セマフォ変数semidが0でないまたはプロセスキューが空の時0である。

[定義 5]　$Sem_1 : M = \{\text{semcom}\}$

Sem_1テスト基準は、セマフォに対する操作の動作が少なくとも1回実行されることを要求する。この場合、測定対象の数はセマフォ命令の数、すなわち、セマフォに対する操作の2倍であるので、テスト基準の複雑さは$O(n)$である。

次に、2つのセマフォ命令が連続して実行された時の動作に基づくSem_2テスト基準を定義する。

[定義 6]　$Sem_2 : M = \{< \mathrm{semcom}_1, \mathrm{semcom}_2 >\}$
ただし＜ ＞は順序対を表す。

＜ $\mathrm{semcom}_1, \mathrm{semcom}_2$ ＞をセマフォ順序対と呼ぶ。セマフォ順序対は、任意のセマフォ命令の対としているので、実際には、実行できない順序対が存在する可能性がある。

Sem_2 テスト基準の複雑さは、$O(n^2)$ である。

[定義 7]　$Sem_\infty : M = \{< \mathrm{semcom}_1, \mathrm{semcom}_2, \cdots >\}$

プログラムに繰り返しが存在する場合、セマフォの実行列は、一般に無限個になるので、このテスト基準を満足する (被覆率を 100% にする) ことは、不可能である。しかしながら、セマフォ命令の全ての実行列を含んでいるので、プロセス間の動機がセマフォのみで行なわれている場合には、このテスト基準が満たされると、デッドロックやライブロックなどの同期誤りを発見できる。

4. 議論

4.1 測定方法

前節で定義したセマフォテスト基準を満たしているか否かを調べるためには、プログラムを実行した時に発生する事象を計測する必要がある。計測は、以下のように探針の挿入によって行なう。*semop* ではそれが P 命令の働きをするのか、V 命令の働きをするのか分かりにくいので、横にコメントを書いて PV の区別をしやすいようにした。

```
loop
    semop(semid);     /*P*/            ....(1)
      {critical section}
    semop(semid);     /*V*/            ....(2)
end loop;
```

プログラムが上のような形態の場合、P 命令 (1)V 命令 (2) について以下の様に、測定用の探針 (probe) を挿入する。

```
loop
    semop(check);     /* P(check) */
      (* semid についての記録を取る *)
    semop(semid);     /* P(semid) */      ....(1)
    semop(check);     /* V(check) */
      {critical section}
    semop(check);     /* P(check) */
      (* プロセスキューについての記録を取る *)
    semop(semid);     /* V(semid) */      ....(2)
    semop(check);     /* V(check) */
end loop;
```

このようにプログラムを変換し、探針を挿入したプログラムを実行すれば、被覆率を計測できる。しかし、変換されたプログラムは探針の影響により元のプログラムと動作が異なることがあり得る。しかし、元のプログラムで有り得ない動作がプログラム変換により起きなければ問題はない。その場合、プログラム変換された後の動作が元のプログラムの動作の一つとなるので変換前の動作をテストで実現したことになる。

4.2 包含関係

[定義 8] 包含関係
Cr_1 テスト基準を満足すれば、必ず Cr_2 テスト基準を満足する場合、Cr_1 が Cr_2 を包含するといい、これを $Cr_1 \supset Cr_2$ で表す。

これを用い、セマフォ順序対テスト基準の包含関係を表すと、次のようになる。

$$Sem_{i+1} \supset Sem_i$$

ただし、Sem_i $(i > 1)$ は測定対象事象として、長さ i のセマフォ命令の実行列をとった場合のテスト基準を表す。

4.3 従来のテスト基準との関係

逐次処理プログラムにおいて用いられているテスト基準として C_0, C_1, $C_2, \cdots$, C_∞ テスト基準がある。C_0 は **All-nodes** 基準、C_1 は **All-edge** 基準、C_∞ は **All-path** 基準とも呼ばれる。

C_0 テスト基準は、全ての **node**、つまりプログラム中の全ての文を少なくとも 1 回実行するというテスト基準である。全ての文が実行されれば、全てのセマフォ命令も実行されることになる。よって $Sem_0 \subset C_0$ という包含関係が成り立つ。

Sem_1 は並行動作を含んでいるので C_1 とは比較不可能である。

5. おわりに

C 言語と UNIX オペレーティングシステムを用いた並行処理プログラムについて、セマフォテスト基準を定義した。

言語仕様でなく、オペレーティングシステムの機能を利用して並行処理を実現している並行処理プログラムにおいては、オペレーティングシステムの機能を提供するライブラリを分析することによって、テスト十分性を評価できることがわかった。

また、UNIX において、プロセス間通信機能としてはセマフォの他にも以下のようなものが有る [3]。

i) パイプ
ii) FIFO あるいは名前付きパイプ
iii) レコードロック
iv) メッセージ伝達
v) 共有メモリ
vi) ソケット

i) 〜 v) までは SYSTEM V の機能であるけれども、6 番目のソケットは UNIX 4.3BSD に備わっている機能である。これらの機能に対するテスト法を考察する必要がある。

レコードのロック、アンロックを P,V 命令に対応させ、ロックされるレコード (ファイル) に対する操作をクリティカルセクションと考えれば、同様のテスト基準がレコードロック機能にも適用できる。

メッセージ伝達や共有メモリ、ソケットを用いた並行処理プログラムについては、テスト充分性評価法 [1, 2] について今回と同様の考察を行なう必要がある。

参考文献

[1] 古川 善吾, 牛島 和夫 : ランデブー通路を用いた Ada 並行処理プログラムのテスト十分性評価, 電子情報通信学会論文誌, D-I Vol.J75-D-I No.5, pp.288-296, 1992.

[2] 有村 耕治, 古川 善吾, 牛島 和夫 : 並行プログラムにおける広域データフローテスト基準の提案, 情報処理学会 第 40 回 (平成 2 年後期) 全国大会講演論文集, pp.1008-1009, 1990.

[3] Keith Haviland, Ben Salama.,(訳 玉井 浩) : UNIX システムプログラミング, サイエンス社, 1991.

[4] Dijkstra,E.W., : The structure of the T. H. E. multiprogramming system, *Comm.* ACM, **11**, pp.341-346,1968.