プログラミング　11－8
(1997. 1. 23)

予約付きセマフォによるデッドロックの回避

孫　志太*　　大原　茂之*　　澤田　勉**

*東海大学工学部電子工学科
**エルグ株式会社

〒259-12　神奈川県平塚市北金目1117
0463－58－1211　内　4107
sunzt@keyaki.cc.u-tokai.ac.jp

あらまし　再利用可能なリソースの割り付けにおけるデッドロックを回避するために，予約付きセマフォ同期メカニズムを定義し，資源を効率的に利用可能とするデッドロック回避方法を提案する．予約付きセマフォは2つのセマフォで構成され，それぞれリソースの獲得状況と予約状況を管理する．タスクのリソースの獲得，予約および解放は予約付きセマフォにおけるＰＲ，ＶＲ操作より行い，リソースの排他的利用とデッドロック回避が同時に実現される．ＰＲ操作がコストO（m），ＶＲ操作がコストO（1）で実現すると同時に，デッドロックを起こさない限り，タスクに予約されても，他のタスクが利用可能にすることによって，リソースの利用率の向上が期待できる．さらに，バイナリ予約付きセマフォ同期メカニズムを示した．

キーワード　デッドロック回避，予約付きセマフォ，ＰＲ操作，ＶＲ操作

Reservation-Extended Semaphore: A Synchronization Mechanism for Deadlock Avoidance

SUN Zhitai*　OHARA Shigeyuki*　SAWADA Tsutomu**

*Department of Electronics, Faculty of Engineering, Tokai University
**ERG Co., LTD.

Kitakaname 1117, Hiratsuka City, Kanagawa Prev., 259-12 Japan
phone:0463-58-1211(Ext.4107)
e-mail:sunzt@keyaki.cc.u.tokai.ac.jp

Abstract　Reservation-Extended semaphore, a new synchronization mechanism is proposed to avoid deadlocks in reusable resources allocation. A reservation-extended semaphore is constituted by two semaphores to manage resources acquired situation and reserved state respectively. To avoid deadlocks, tasks must access the reservation-extended semaphore by using PR and VR operations to acquire, reserve or release a resource. The executing cost are $O(m)$ for PR and $O(1)$ for VR. The mechanism can also grant a task to acquire a reserved resource with deadlock free. So, the reservation-extended semaphore is an effective synchronization mechanism for resources allocation. Moreover, we have also shown the binary reservation-extended semaphore synchronization mechanism.

key words　Deadlock Avoidance, Reservation-Extended Semaphore, PR Operation, VR Operation

1. まえがき

デッドロックは，２つ以上のタスクが互いに相手が保持しているリソースを解放することを待つときに起きる[1]～[13]．この問題を解決するためには，次の３つの方法がある[4]．第１はシステム設計時にタスクの動作を制約し，デッドロックを未然に防止する方法である．しかし，この方法はリソースの利用率の低下を招きかねない．第２はデッドロックを検出して回復する方法である．この方法の場合，リソースを有効に利用できるが，リソースを１つ獲得するごとに，デッドロックの発生をチェックしなければならない．そのため，O（タスク数×リソース種類）のコストが必要となる．第３は，デッドロックの発生可能性を予見し，その発生を回避する方法である．この方法には，予めタスクごとのリソースの種類と必要量を宣言し，必要とするすべてのリソースを一括して獲得する手法として，multiple P/V[3]，general P/V[5]などがある．また，リソースを必要となる時点で，デッドロックが発生する可能性をチェックしながら獲得する手法として，銀行家アルゴリズム[1]がある．このアルゴリズムでリソースを有効に利用できるが，デッドロックの発生可能性を検出するのにかかるコストはO（タスク数×リソース種類）となる．一方，一括して獲得する手法はリソースを簡単に管理できる反面，獲得されたリソースが利用されていない間でも，他のタスクからの利用は不可能になり，リソースの利用率を低下させ，システム全体の効率を低下させてしまう．

われわれは，タスクに獲得されたリソースの利用状況を使用中と未使用に分けて考えることとした．タスクに獲得されながら，未使用の再利用可能なリソースが，デッドロックにならないかぎり他のタスクに利用されてもよい予約付きセマフォ同期メカニズムを導入する．予約付きセマフォは，２つのセマフォで構成され，第１のセマフォが利用中のリソースの数を管理し，第２のセマフォが予約状況を管理する．この予約付きセマフォ同期メカニズムを用いることによって，デッドロックを回避するとともに，効率的にリソースを利用することはできる．予約付きセマフォを利用するときのオーバーヘッドはO（タスクが必要とするリソースの種類）である．

以下，第２章では，予約付きセマフォとそのプリミティヴ的操作ＰＲとＶＲについて定義し，例題を示しながらＰＲ操作とＶＲ操作の利用方法について述べる．第３章では，予約付きセマフォを利用することによって，デッドロックが回避できることを証明する．第４章では，効率的なバイナリ予約付きセマフォについて述べる．最後に，予約付きセマフォメカニズムによるデッドロック回避について総括し，これからの研究課題を示す．

2. 予約付きセマフォのメカニズム

この章において，予約付きセマフォとその上における操作について述べる．

2．1　予約付きセマフォ

タスクがリソースを利用するとき，獲得と予約，そして，解放が考えられる．獲得はタスクがリソースを排他的に利用することを保証し，予約はタスクが将来獲得するリソースを確保した上で，デッドロックにならない限り，他のタスクの利用を許す．解放はタスクが利用済みのリソースを他のタスクに利用できるようにする．このような制御を可能にするために，予約付きセマフォを次のように定義する．

［定義１］　予約付きセマフォとは，（u__s，r__s）の２項組である．ただし，u__s，r__sは整数型の共有変数であり，初期値はu__s＝r__s＝n，nは正整数である．

［定義おわり］

タスクが予約付きセマフォをアクセスするための手段としては，ＰＲ操作とＶＲ操作を定義する．

2．2　ＰＲ操作

［定義２］　ＰＲ操作とはPR(u_s, n)または$PR(r_s_1, n_1, r_s_2, n_2, \ldots, r_s_m, n_m)$のことであり，予約付きセマフォにおける分割不可な手続きである．

PR(u_s, n)（ただし，n≦u_sの初期値）に対しては，IF (u_s>=n) THEN u_s=u_s-nとする．

$PR(r_s_1, n_1, r_s_2, n_2, \ldots, r_s_m, n_m)$に対しては，

$$
\begin{aligned}
&\text{IF } (r_s_1>=n_1)\wedge\ldots\wedge(r_s_{m-1}>=n_{m-1})\\
&\quad\vee(r_s_1>=n_1)\wedge\ldots\wedge(r_s_{m-2}>=n_{m-2})\wedge(r_s_m>=n_m)\\
&\quad\vee\ldots\\
&\quad\vee(r_s_1>=n_1)\wedge(r_s_3>=n_3)\ldots\wedge(r_s_m>=n_m)\\
&\quad\vee(r_s_{m-2}>=n_{m-2})\wedge\ldots\wedge(r_s_m>=n_m)\\
&\text{THEN}\\
&\quad r_s_1 = r_s_1 - n_1;\\
&\quad r_s_2 = r_s_2 - n_2;\\
&\quad \ldots\ldots;\\
&\quad r_s_m = r_s_m - n_m
\end{aligned}
$$

とする．ここで，記号∧，∨でそれぞれ論理積と論理

和を表す．m（$2 \leqq m$）は予約付きセマフォの数であり，n_iは各予約セマフォの減少量である．ただし，$1 \leqq i \leqq m$，$n_i \leqq$ r__s$_i$の初期値である．

［定義おわり］

タスクがPR(u_s, n)によって，リソースを獲得し，PR(r_s$_1$, n$_1$, r_s$_2$, n$_2$, . . . , r_s$_m$, n$_m$)によって，リソースを予約する．定義2より，次のPRアルゴリズムが得られる．ただし，①～⑦は説明のために付けたラベルである．

［PRアルゴリズム］

```
  PR(u_s,n,r_s1,n1,...,r_sm,nm)
  {
①    if (u_s>=n)
②      u_s -= n;
③    else{  if (m <= 1) BLOCK;
         else {
④         for(i=1,c=0;i<=m;i++) if (r_si<ni) c++;
⑤         if ( c<2 )
⑥             for ( i=1; i<=m; i++)   r_si -=ni;
⑦         else BLOCK;
         }
     }
  }
```

PRアルゴリズムにおいて，①では許可条件u_s>=nをチェックし，満足していれば②でu__sに対して減量を行う．③では操作対象がr__sであるかu__sであるかを判断する．操作対象がu__sであれば，条件が満足していないときの処理を行う．④以下は，操作対象がr__sであるときの処理を行う．まず④ではカウントが0以下になるr__sの数を算出し，⑤では定義2におけるr__sの状態より構成される許可条件をチェックする．許可条件が満足されたとき，⑥でr__sから要求量を減らす．許可条件が満足していないとき，⑦でタスクの要求を拒否する．

PRアルゴリズムにおいて，操作対象がu__sであれば，u__sに対する処理が1ステップで完了される．操作対象がr__sであれば，r__sに対する処理がタスクの要求するリソースの種類m（>1）に比例する．すなわち，PR操作にかかるコストは最悪の場合が$\mathcal{O}$(m)である．

2．3　VR操作

タスクがリソースを解放する操作として，次のようにVR操作を定義する．

［定義3］　VR操作とはVR(u_s, n)またはVR(u_s, n, r_s)のことであり，予約付きセマフォにおける分割不可な処理手続きである．

VR(u_s, n)に対しては，u_s=u_s+nとする．

VR(u_s, n, r_s)に対しては，u_s=u_s+n, r_s=r_s+nとする．

ここで，$n \leqq$ u__sの初期値である．

［定義おわり］

定義3より，次のVRアルゴリズムが得られる．

［VRアルゴリズム］

```
VR(u_s,n,r_s)
{
   if ( r_s == NULL )  u_s = u_s + n;
   else  {
       u_s = u_s + n;
       r_s = u_s + n;
   }
}
```

VR操作は，操作対象がu__sのみであるとき，u__sに対する処理が1ステップで完了する．操作対象がu__sとr__s両方であるとき，u__sとr__sに対する処理がそれぞれ1ステップで完了する．よって，VR操作にかかるコストは$\mathcal{O}$（1）である．

3．予約付きセマフォによるデッドロックの回避

3．1　予約付きセマフォによるリソース管理

前章で述べたように，タスクのリソースに対する利用は，予約付きセマフォ（u__s，r__s）に対するPR操作，VR操作によって実現される．例えば，タスクtが獲得したリソースの集合をt（u__s）とし，tが予約したリソースの集合をt（r__s）とすれば，タスクtが利用するすべてのリソースの集合は，t（u__s）とt（r__s）の和集合となる．PR操作の定義から，r∈t（r__s）のとき，rのr__sが0以下となる可能性がある．このとき，rを予約超過リソースとよぶ．t（r__s）に含まれるすべての予約超過リソースをtの予約超過リソース集合とよび，OVER（t）と記す．

タスクtがリソースrを獲得する方法には，直接獲得と予約後獲得がある．tがrを予約せずに獲得するとき，直接獲得といい，tがrを予約してから，獲得するとき，予約後獲得という．

タスクがPR操作を用いて，リソースを獲得もしく

は予約するときは，次の規則に従わなけらばならない．

```
task()
{ ......
  PR(u_s, n);
  ......
}
```

(a) Request one resource type

```
task()
{ ......
  PR(r_s1, n1, r_s2, n2);
  ......
  PR(u_s1, n1);
  ......
  PR(u_s2, n2);
  ......
}
```

(b) Request two resource types

図１　ＰＲ操作によるリソースの要求
Fig. 1 Resources request by using PR operation

［規則１］

① タスクが２種類以上のリソースを要求するとき，各リソースを獲得する前に，ＰＲ操作によりすべてのリソースを予約しなければならない．

② リソースを獲得するとき，１回のＰＲ操作で１種類のリソースを必要量（予約量）だけで獲得しなければならない．

③ タスクが１種類のリソースのみを利用するとき，直接獲得しなければならない．

［規則おわり］

規則１の①は，タスクが１つのクリティカル区域において，数回に分けてリソースを予約することを禁ずる．この規則を従わないと，ＰＲ操作に関する定義２の判断条件が崩れるために，デッドロックの回避を保証できない．②は，タスクが同種類のリソースを利用するとき，数回に分けて獲得することを禁ずる．この規則は，２つ以上のタスクが同じ種類のタスクを利用するとき，互いに相手が獲得しているリソースを待つことを回避する．③は，タスクが１種類のみのリソースを予約することを禁ずる．この規則を守らないと，タスクは永久にリソースを利用することができなくなる．

［例１］タスクが規則１に従い，ＰＲ操作によって，リソースを要求する例を図１に示した．

［例おわり］

３．２　デッドロック回避のメカニズム

予約付きセマフォを用いることによって，デッドロックにならないことは，次の２つのケースについて証明できればよい．

ケース①：タスクが１種類のリソースのみ利用し，一括でリソースの必要量を直接獲得する場合である．このとき，予約付きセマフォ同期メカニズムは，リソースの利用可能量が許可されたすべてのタスクに満足できることを保証する．

ケース②：タスクが複数種類のリソースを獲得する場合である．このとき，デッドロックにならないことは，タスクが複数種類のリソースを予約して，そして，予約後獲得を行うため，予約が許可された任意の２つのタスク t_i，t_j の予約超過リソース集合の積集合

$$OVER(t_i) \cap OVER(t_j)$$

に属する要素が２以下である．

証明①：すべてのタスクが１種類のリソースのみ利用する場合，ＰＲ操作の利用規則より，必要とするリソース量を一括にして直接獲得を要求するため，一部のリソースを獲得しながら，他のタスクによるリソースの解放を待つ必要がない．つまり，この場合，デッドロックにならない．

証明②：１つのタスクのみ，複数種類のリソースを利用する場合，各種類のリソースを管理する予約セマフォが，このタスクのみに利用される．このため，予約要求が許可される．そして，予約後獲得でリソースを１種類ずつ獲得するため，前述の証明①と同じになり，デッドロックにならない．

証明③：複数のタスクが複数種類のリソースを利用するときについて数学的帰納法の手法で証明する．

まず，図２に示したように，２つのタスク t_1，t_2 が予約付きセマフォで管理された複数種類のリソースを要求する場合，t_1 の予約超過集合を $OVER(t_1)$

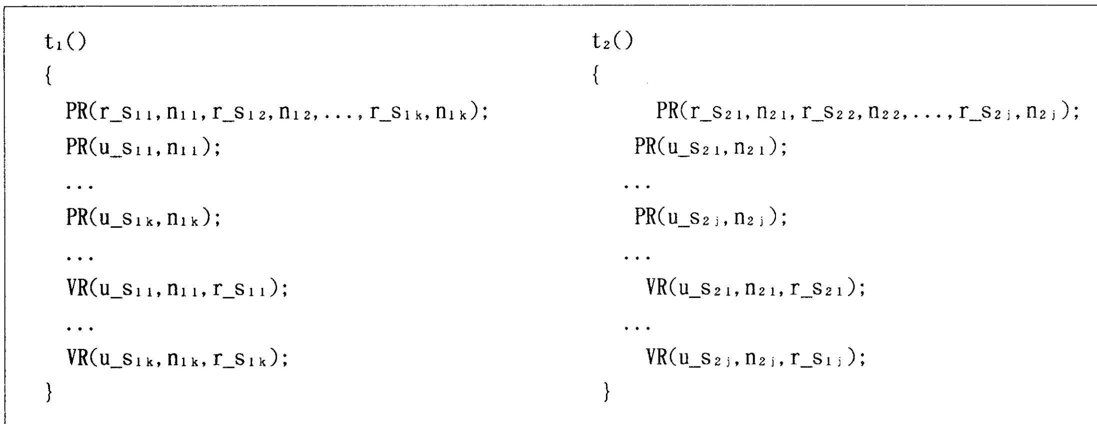

図２　２つのタスクによる複数種類のリソースの利用
Fig. 2 Two tasks request multiple resources

とし，t_2の予約超過集合をＯＶＥＲ（t_2）とする．

もし，ＯＶＥＲ（t_1）∩ＯＶＥＲ（t_2）＝∅であれば，t_1とt_2がそれぞれの処理シーケンスに入ることができる．デッドロックになる必要条件を満たさないため，デッドロックにはならない．

もし，タスクがリソースを予約するとき，ＰＲ操作により＃（ＯＶＥＲ（t_1）∩ＯＶＥＲ（t_2））＞１と判定すれば，タイミング的に後で要求するタスクはブロックされ，要求するすべてのリソースを取らずに待ち状態に入る．つまり，１つのタスクの予約リソース集合ｔ（ｒ＿ｓ）は空となる．よって，ＯＶＥＲ（t_1）∩ＯＶＥＲ（t_2）＝∅となり，デッドロックにはならない．

もし，＃（ＯＶＥＲ（t_1）∩ＯＶＥＲ（t_2（Ｒ））＝１であれば，t_1とt_2ともに利用するリソース種類の内，１種類のリソースが予約超過となる．ＰＲ操作の定義より，一方のタスクがこのリソースを予約後獲得するとき，必要量のリソースを獲得できなければ，他のタスクが利用終了まで待つ．このとき，どれか１つのタスクの要求が必ず満足できるため，デッドロックにならない．

仮にｎ－１個のタスクが複数種類のリソースを要求するときデッドロックが発生しないとする．さらに，ｎ番目のタスクが複数のリソースを要求する場合，各タスクの予約超過集合をそれぞれＯＶＥＲ（t_1），ＯＶＥＲ（t_2），．．．，ＯＶＥＲ（t_{n-1}），ＯＶＥＲ（t_n）とする．

もし

$$\{OVER(t_1)\cup\ldots\cup OVER(t_{n-1})\}\cap OVER(t_n)=\emptyset$$

であれば，各タスクは各々の処理シーケンスに入ることができ，デッドロックにならない．

もし

$$\{OVER(t_1)\cup\ldots\cup OVER(t_{n-1})\}\cap OVER(t_n)\neq\emptyset$$

であれば，２つのケースに分けられる．

ケース１：t_nのＰＲ操作より，集合$\{OVER(t_1)\cup\ldots\cup OVER(t_{n-1})\}\cap OVER(t_n)$の濃度が２以上であると判断された場合，$t_n$がリソースを取らずに待ち状態に入るため，レディ状態にあるタスクがｎ－１のままであり，デッドロックにならない．

ケース２：t_nのＰＲ操作より，集合$\{OVER(t_1)\cup\ldots\cup OVER(t_{n-1})\}\cap OVER(t_n)$の濃度が１であると判断された場合，$t_n$のリソース予約要求が許可される．このとき，$t_n$のＰＲ操作によって，１種類のリソースが予約超過となる．ＰＲの定義より，t_nは必ず予約したリソースの予約後獲得を行い，予約超過となるリソースのみ獲得できない可能性がある．しかし，仮定より，t_1，．．．，t_{n-1}の要求に満足できないリソース種類がこの予約超過リソース以外に存在しない．さもなければ，ケース１になる．規則１より，タスクが１種類のリソースを利用するとき，必要とする量を１回のＰＲ操作で取るため，複数の同種類のリソースを利用時のデッドロックが回避される．故に，このときもデッドロックになることがない．

［例２］３つのタスクt_1，t_2，t_3と３種類のリソースr_1，r_2，r_3から構成されるシステムを図３に示す．r_1，r_2，r_3の個数はそれぞれ３個，２個，２個

```
    t1()                              t2()                              t3()
    {                                 {                                 {
①     PR(r_s3,2,r_s1,3);               PR(r_s1,2,r_s2,2);                PR(r_s2,1,r_s3,1);
      ...;                             ...;                              ...;
②     PR(u_s3,2);                      PR(u_s1,2);                       PR(u_s2,1);
      ...;                             ...;                              ...;
③     PR(u_s1,3);                      PR(u_s2,2);                       PR(u_s3,1);
      ...;                             ...;                              ...;
④     VR(u_s3,2,r_s3);                 VR(u_s1,2,r_s1);                  VR(u_s2,1,r_s2);
⑤     VR(u_s1,3,r_s1);                 VR(u_s2,2,r_s2);                  VR(u_s3,1,r_s3);
    }                                 }                                 }
```

図３　予約セマフォによってデッドロックが回避されたシステム例

Fig.3 A deadlock free system by using reservation-extended semaphores

	τ_0	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9	τ_{10}	τ_{11} time
t_1		①		②		③			④	⑤		
t_2			①		②		③	④	⑤			
t_3					①				②	③	④	⑤

(a) An executing sequence of Fig.3

	τ_0	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9	τ_{10}	τ_{11}
u_s1	3	3	3	3	1	1	1	0	0	3	3	3
r_s1	3	0	-2	-2	-2	-2	-2	0	0	3	3	3
u_s2	2	2	2	2	2	2	0	0	0	1	2	2
r_s2	2	2	0	0	0	0	0	0	0	1	2	2
u_s3	2	2	2	0	0	0	0	0	1	1	1	2
r_s3	2	0	0	0	0	0	0	0	1	1	1	2
					*	*		**	**			

(b) The reservation-extended semaphores response of (a)

図４　図３の実行シーケンス例と予約付きセマフォの変化

Fig.4 An executing sequence and the reservation-extended semaphores responses of Fig.3

である．t_1はr_3を２個，r_1を３個利用する．t_2はr_1を２個，r_2を２個利用する．そして，t_3はr_2を１個，r_3を１個利用する．仮に，t_1が先に実行され，r_3，r_1を予約する．予約セマフォを用いることによって，t_1のPR(u_s1,3)が実行されるまで，t_3が実行されなければ，t_2がリソースを予約し，クリティカル区域にはいることができる．また，t_2が実行されなければ，t_3が任意の時点でリソースを予約し，そのクリティカル区域に入ることができる．このとき，図４（ａ）に示したような実行シーケンスが考えられる．

図４（ａ）の実行シーケンスに対し，予約付きセマフォが図４（ｂ）に示したように変化する．許可されたタスクのリソースを管理する予約付きセマフォのうち，マイナス値を取るのは高々１つである．図４（ｂ）において，タスクの要求が拒否されたタイミングは＊で示してある．τ_4では，デッドロックを回避するため

に，t_3のr_2，r_3に対する予約要求が拒否されている．τ_5では，t_1のr_1に対する予約後獲得要求が拒否されている．拒否されたタスクが再開するタイミングは**で示してある．τ_7でt_2がr_1を解放し，t_1の要求が満たされる．τ_8でt_2がr_2を解放し，t_3のr_2，r_3に対する予約要求が満たされる．

［例終わり］

4．バイナリ予約付きセマフォによるデッドロックの回避

［定義4］バイナリ予約付きセマフォとは，初期値1の予約付きセマフォである．

［定義終わり］

バイナリセマフォにより管理されているリソースが1つであるため，ＰＲ操作のアルゴリズムを次のように簡略化することができる．

［バイナリＰＲアルゴリズム］

```
BPR(u_s1,r_s1,...,r_sm)
{
  if (u_s1>0)
    u_s1 -= 1;
  else {
    if (m<=1) BLOCK;
    else {
      for(i=1,c=0;i<=m;i++) if (r_si<1) c++;
      if (c<2)
          for ( i=1; i<=m; i++)   r_si-=1;
      else BLOCK;
    }
  }
}
```

［アルゴリズム終わり］

同様に，ＶＲ操作のアルゴリズムを次のように簡略化することができる．

［バイナリＶＲアルゴリズム］

```
BVR(u_s,r_s)
{
    if ( r_s == NULL )
        u_s++;
    else {
        u_s++;
        r_s++;
    }
}
```

［アルゴリズム終わり］

バイナリＰＲ操作，バイナリＶＲ操作において，セマフォ内容の増減は1であるため，プログラミングにおいて，レジスタ変数を利用することによって，時間の節約を図ることができる．

5．おわりに

本論文では，予約付きセマフォ同期メカニズムを提案し，デッドロック回避方法を示した．この手法を用いて，最悪O（タスクが必要とするリソースの種類）のコストでデッドロックを回避できることがわかった．さらに，タスクに予約されたリソースでも，デッドロックにならない限り，他のタスクに利用されることが可能になり，multiple PV同期メカニズムよりも，リソースの利用率が向上できることが分かった．現在，予約付きセマフォを用いて，使い捨て型と再利用可能型リソースが共存するシステムに対するデッドロック回避について検討中である．

参考文献

1. Dijkstra,E.W.,(1968)."Co-operating sequential processes". In Programming Languages, Academic Press, New York. pp.43-112.
2. Habermann,A.N.,(July 1969)."Prevention of system deadlocks". Communications of the ACM 12, 7, pp.373-377,385.
3. Patil,S.S.(Feb. 1971). "Limitations and capabilities of Dijkstra's semaphore primitives for coordination amond processes". MIT Project MAC Computation Structure Group Memo 57, MIT, Cambridge, Mass.
4. Coffman,E.G.,Jr.;M.J.Elphick; and A.Shoshani.(1971)."System deadlocks". ACM Computing Surveys 3, 2, pp.67-68.
5. Lipton,R.J.(1974). "A Comparative Study of Models of Parallel Computation". 15th Annual Symposium on Switching and Automata Theory, pp.145-155.
6. Frailey,D.J.,(May 1973)."A practical approach to managing resources and avoiding deadlocks." Communications of the ACM 16,5,pp.323-329.
7. Gligor,V.,and S.Shattuch.(Sep. 1980). "On deadlock detection in distributed Systems." IEEE Transactions on Software Engineering SE-6, pp.435-440.

8. Holt, R. C. (Jan. 1971). "Comments on Prevention of system deadlocks." Communications of the ACM 14, 1, pp. 36-38.
9. King, P. F., and A. J. Collmeyer. (1973). "Database sharing--an efficient mechanism for supporting concurrent processes." Proceedings of the AFIPS National Computer Conference 42, pp. 271-275.
10. Lomet, D. B. (1980). "Subsystems of processes with deadlock avoidance." IEEE Transactions on Software Engineering SE-6, pp. 297-304.
11. Murphy, J. E. (1968). "Resource alocation with interlock detection in a multi-task system." Proceedings of the AFIPS Computer Conference 33, pp. 1169-1176.
12. Parnas, D. L. and A. N. Habermaan. (1972). "Comment on a deadlock prevention method." Communications of the ACM 15, 9, pp. 840-841(with reply by R. C. Holt).
13. Rypka, D. J., and A. P. Luciado. (1979). "Deadlock detection and avoidance for shared logical resources." IEEE Transactions on Software Engineering SE-5, pp. 465-471.
14. Shoshani, A., and A. J. Bernstein. (Nov. 1969). "Synchronization in a parallel-accessed database." Communications of the ACM 12, 11, pp. 604-607.