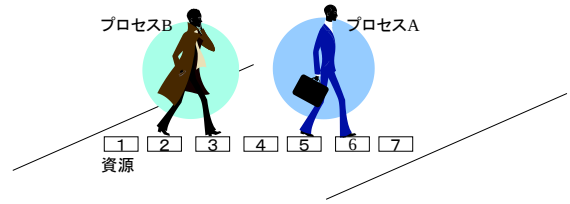
第7回　デッドロック（2）

デッドロックの防止

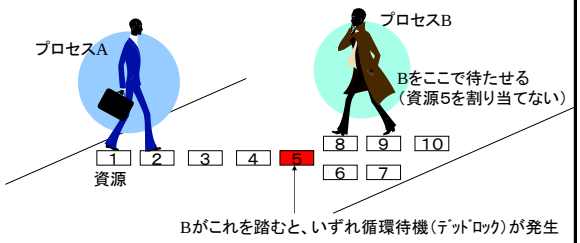
一方通行にし、衝突が起こりえないようにする
資源に番号を付け、各プロセスは、番号順でしか資源を確保しない。



プロセスBは、1、2、3、4、5、6、7の順で
資源を使用するようにプログラムを修正

デッドロックの回避

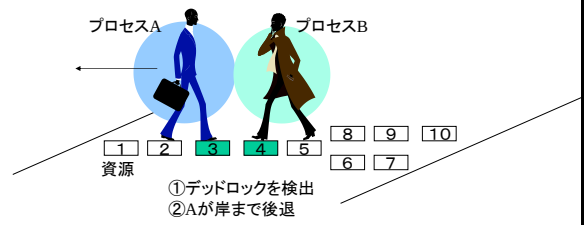
目の前の踏み石だけでなく、向こう岸まで見て、衝突を避ける
資源要求時、全ﾌﾟﾛｾｽの資源の使用状態を検査。危険があれば割り当てない。



Aは、資源1、2、3、4、5、6、7を使用
Bは、資源10、9、8、5、4、3、2、1を使用

デッドロックの検出と回復

自由に渡らせるが、衝突したら一方が岸（出発点）に後退
ﾌﾟﾛｾｽを強制終了し、ﾁｪｯｸﾎﾟｲﾝﾄ（同期点）から再開させる（ロールバック）



解決法1：ﾃﾞｯﾄﾞﾛｯｸの防止

ﾃﾞｯﾄﾞﾛｯｸの必要条件が成立しないようにする（何れか1つが否定できればよい）

必要条件	必要条件の成立を防止するための方法	評価
相互排除	全ての資源を共用できるようにする	実現不可能　××
確保と待機	資源を確保したまま資源待ちしない ・必要な資源を実行開始時に全て要求 ・資源待ち時は、確保済み資源を解放	処理が複雑△ ・資源の使用率低下 ・再取得処理が必要
横取り不能	待機中ﾌﾟﾛｾｽから、資源を横取り （横取りした資源は、後で再割り当て）	制御が複雑× ・使用途中の資源の扱い
循環待機	資源型に順序を付け、アプリケーションには、若番の資源型から順に要求するようにプログラミングさせる	○ 他の条件を否定するのは困難（不可能）

ﾃﾞｯﾄﾞﾛｯｸの防止：このような制限をつけて循環待機が成立しないようにする
（老番の資源を先に使用したい時は、若番の資源も先に確保してしまう）

問題点：APのﾌﾟﾛｸﾞﾗﾐﾝｸﾞに制約が付く。資源の使用効率が非常に悪い。

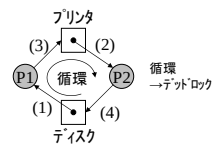
デッドロック防止の例

ﾃﾞｯﾄﾞﾛｯｸ防止を行わない場合

P1：
①磁気ﾃﾞｨｽｸ要求
②ﾌﾟﾘﾝﾀ要求
③磁気ﾃﾞｨｽｸ解放
④ﾌﾟﾘﾝﾀ解放

P2：
⑤ﾌﾟﾘﾝﾀ要求
⑥磁気ﾃﾞｨｽｸ要求
⑦ﾌﾟﾘﾝﾀ解放
⑧磁気ﾃﾞｨｽｸ解放

①⑤②⑥の順で処理されると
(1)P1：①磁気ディスク要求→割当て
(2)P2：⑤プリンタ要求→割当て
(3)P2：②プリンタ要求→待機
(4)P1：⑥磁気ディスク要求→待機



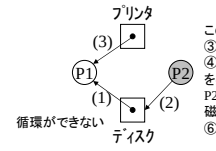
ﾃﾞｯﾄﾞﾛｯｸ防止を行う場合

以下の順序で資源を要求するように決め，
P2のプログラムを変更

資源型	順序
磁気ディスク	1
プリンタ	2

P2：
⑤磁気ﾃﾞｨｽｸ要求
⑥ﾌﾟﾘﾝﾀ要求
⑦磁気ﾃﾞｨｽｸ解放
⑧ﾌﾟﾘﾝﾀ解放

①⑤②の順で処理
(1)P1：①磁気ディスク要求→割当て
(2)P2：⑤磁気ディスク要求→待機
(3)P1：②プリンタ要求→割当て
（P2は待機状態なので，⑥は実行されない）



この後P1は
③磁気ﾃﾞｨｽｸ解放
④ﾌﾟﾘﾝﾀ解放
を実行．
P2は再開して，
磁気ﾃﾞｨｽｸが割当てられ
⑥ﾌﾟﾘﾝﾀ要求　以降を実行

デッドロック防止（補足）

- 前スライドにおける決められた要求の順番
- ①磁気ﾃﾞｨｽｸ要求→②ﾌﾟﾘﾝﾀ要求

- P1：①磁気ﾃﾞｨｽｸ要求→③ﾌﾟﾘﾝﾀ要求→磁気ﾃﾞｨｽｸ解放→ﾌﾟﾘﾝﾀ解放
- （1）順番を守らない場合
- P2（修正前）：
- ①ﾌﾟﾘﾝﾀ要求→②磁気ﾃﾞｨｽｸ要求→ﾌﾟﾘﾝﾀ解放→磁気ﾃﾞｨｽｸ解放

- （2）順番を守るように変更
- P2（修正後）：
- ①磁気ﾃﾞｨｽｸ要求→②ﾌﾟﾘﾝﾀ要求→ﾌﾟﾘﾝﾀ解放→磁気ﾃﾞｨｽｸ解放

- 要求の順番を合わせると、循環待機が発生しなくなる

解決法2：デッドロックの回避

- ﾌﾟﾛｾｽが資源を要求した時に、その資源を割り当てるとデッドロックが起こり得るかどうかを判定。起こり得る場合は、資源が空いていても割り当てない。
- 判定ｱﾙｺﾞﾘｽﾞﾑ（銀行家アルゴリズム）の考え方
  - 資源の割り当て状態を管理（各ﾌﾟﾛｾｽへの割当て数、空き数）
  - 各ﾌﾟﾛｾｽは要求する資源の最大数を申告。
  - 資源を割当てても安全な順序（空き資源を使い全ﾌﾟﾛｾｽが終了できる順序）が存在するかどうかを判定
    - 存在：ﾃﾞｯﾄﾞﾛｯｸが起こらないので資源を割り当てる
    - 存在しない：要求を待たせる
- 問題点
  - 処理負荷が大：要求の都度、判定ｱﾙｺﾞﾘｽﾞﾑの実行が必要
  - 資源の最大数の宣言が必要（予測できない場合もある）
  - 資源の使用効率が悪い

銀行家ｱﾙｺﾞﾘｽﾞﾑ（資源型が1種類の場合）

- ﾌﾟﾛｾｽPiへの割り当て済み資源数Ai、Piの最大使用資源数Mi
- システム全体の残り資源数：W
- 今、ﾌﾟﾛｾｽPjが資源をRj個要求した。資源を割り当てて良いか？

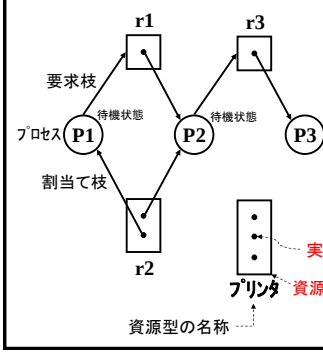
- W＜Rjの場合（資源が足らない）
  - Pjの要求を待たせる。
- W≧Rjの場合（資源は足りている）
  - Aj＝Aj＋Rj、W＝W－Rj（Pjに仮割り当て）。安全な順序を探す。
    - （Mi－Ai）≦Wが成立するﾌﾟﾛｾｽPiを探す
    - Fi＝終了可能
    - W＝W＋Ai

    （繰り返し）
  - 安全な順序あり：全ﾌﾟﾛｾｽが、Fi＝終了可能の場合
    - Pjが要求した資源を割り当て。
  - 安全な順序が無い：Fi≠終了可能　というプロセスがある場合
    - 変数を元に戻し、Pjの要求を待たせる

前回のスライド

重要：資源割り当てグラフ

デッドロックの発生を検出するために、資源の割り当て状況を正確に記述



要求枝：ﾌﾟﾛｾｽ→資源型
割当て枝：資源型の実体→ﾌﾟﾛｾｽ

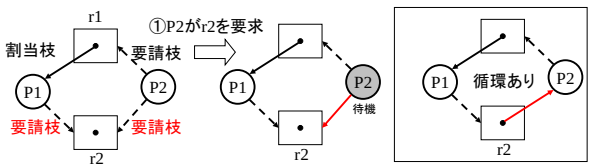
有向グラフ：枝に方向性がある

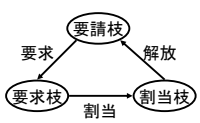
資源を要求すると要求枝ができる
資源を割り当てると要求枝は割当て枝に変わる
（割当て済み資源を待つ間は，待機状態）
資源を解放すると割当て枝が消える

ﾌﾟﾛｾｽの資源要求に基づく資源割り当てグラフが作図できるようになること

ﾃﾞｯﾄﾞﾛｯｸの回避

資源型の実体が全て1個の場合、循環待機が起こり得るかどうかで判定可能
資源割当てグラフの初期状態に以下の要請枝を追加
ﾌﾟﾛｾｽがﾌﾟﾛｸﾞﾗﾑ中でその資源を使う場合（プロセス→資源型の方向の破線矢印）

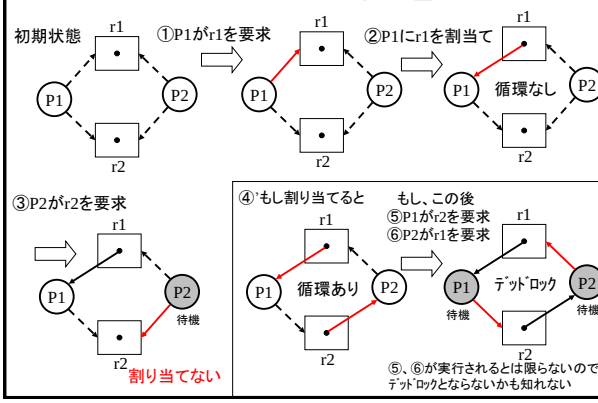




r2を割当てると、要請枝を含めて、グラフに循環。
未使用だが、割り当てない（要求ﾌﾟﾛｾｽは待機状態）。
循環ができなければ割当てる

資源を要求すると要請枝は要求枝に変わる
資源を割り当てると要求枝は割当て枝に変わる
資源を解放すると割当て枝は要請枝に変わる

ﾃﾞｯﾄﾞﾛｯｸの回避



解決法3：ﾃﾞｯﾄﾞﾛｯｸの検出と回復

- ﾌﾟﾛｾｽが資源を要求時、資源に空きがあれば無条件に割当てる
- デッドロック検出アルゴリズムを定期的に起動

- 検出ｱﾙｺﾞﾘｽﾞﾑの考え方
  - 資源の使用状況を管理（各ﾌﾟﾛｾｽへの割当て数、割り当て待ち数、システム全体の空き資源数）
  - 空きの資源を使い、全ﾌﾟﾛｾｽの割り当て待ち資源を満足する順序が存在するかどうかを判定する。
    - 現在、割当て待ち資源を与えられれば、プロセスは終了できる可能性がある。（後で、ﾃﾞｯﾄﾞﾛｯｸになるかも知れないが、今は心配しない）
    - 要求を満足するﾌﾟﾛｾｽへの割当て資源は、空き数として加算（要求が満たされれば終了でき、確保中の資源を返却できる可能性がある）
  - 上記の順序が存在しなければ、デッドロック

- デッドロック検出時
  - 該当ﾌﾟﾛｾｽをロールバック（強制終了し、ﾁｪｯｸﾎﾟｲﾝﾄから再開）
  - または　確保済み資源を横取り

検出ｱﾙｺﾞﾘｽﾞﾑ（資源型が1種類の場合）

- プロセスPiへの割り当て済み資源数：Ai、Piの割り当て待ち資源数Ri
- システム全体の残り資源数：W

- Ai＝0のPiについて、Fi＝非ﾃﾞｯﾄﾞﾛｯｸ(注1)
- Ai＞0のﾌﾟﾛｾｽについて、要求が満たせる順序を探す。
  - Ri≦Wが成立するﾌﾟﾛｾｽPiを探す(注2)。
  - Fi＝非ﾃﾞｯﾄﾞﾛｯｸ
  - W＝W＋Ri(注3)

  } 繰り返し
- 全ﾌﾟﾛｾｽが、Fi＝非ﾃﾞｯﾄﾞﾛｯｸならば、ﾃﾞｯﾄﾞﾛｯｸは無い。
- Fi≠非ﾃﾞｯﾄﾞﾛｯｸのﾌﾟﾛｾｽはﾃﾞｯﾄﾞﾛｯｸ。

- 注1：ﾃﾞｯﾄﾞﾛｯｸの必要条件（確保と待機）を満足しない。
- 注2：残りの資源で要求が満足できればﾃﾞｯﾄﾞﾛｯｸではない。
- 注3：要求が満たされるﾌﾟﾛｾｽは終了でき、確保中の資源を返却できる可能性がある。

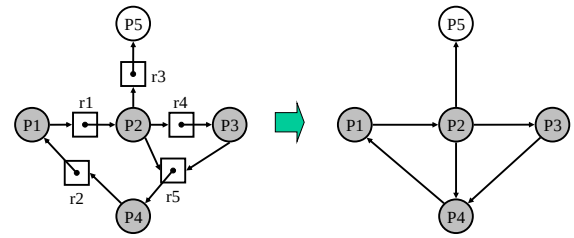
検出アルゴリズム（補足）

- 総資源数8、残り資源数0
- P1：資源を2個確保。4個要求（待ち合せ）　→④非ﾃﾞｯﾄﾞﾛｯｸ
- P2：資源を2個確保　→②非ﾃﾞｯﾄﾞﾛｯｸ
- P3：資源を2個確保。2個要求（待ち合せ）　→③非ﾃﾞｯﾄﾞﾛｯｸ
- P4：資源確保なし。　→①非ﾃﾞｯﾄﾞﾛｯｸ
- P2が終了して返した資源で、P3の要求が満たされ、P3が終了して返した資源でP4の要求も満たされる

- 総資源数8、残り資源数1
- P1：資源を2個確保。4個要求（待ち合せ）　→ﾃﾞｯﾄﾞﾛｯｸ
- P2：資源を2個確保　→②非ﾃﾞｯﾄﾞﾛｯｸ
- P3：資源を3個確保。4個要求（待ち合せ）　→ﾃﾞｯﾄﾞﾛｯｸ
- P4：資源確保なし。　→①非ﾃﾞｯﾄﾞﾛｯｸ
- P2が終了して返した資源では、P1、P3の要求を満たさない

重要：待ち合わせｸﾞﾗﾌ

実体の数が1個の場合のﾃﾞｯﾄﾞﾛｯｸ検出アルゴリズム
資源割当てｸﾞﾗﾌ：循環があればﾃﾞｯﾄﾞﾛｯｸ
（循環の検出さえできれば良い）
待ち合せｸﾞﾗﾌ：ﾌﾟﾛｾｽ間の待ちの関係を示す（資源割り当てｸﾞﾗﾌを簡略化）
待ち合せグラフに循環があればﾃﾞｯﾄﾞﾛｯｸ



資源割当てグラフ　　待ち合せグラフ

検出と回復における考慮要因

- 検出ｱﾙｺﾞﾘｽﾞﾑの呼び出し契機
  - デッドロックの頻度
  - デッドロック発生時に影響を受けるプロセス数
- ﾃﾞｯﾄﾞﾛｯｸからの回復方法
  - プロセスの終了（その結果資源も横取り）
    - デッドロック中の全プロセス
    - 循環がなくなるまで、順番に終了（誰を選ぶか）
    - ファイル書込み中の対処、ﾌﾟﾘﾝﾀの再設定等も必要
  - 資源の横取り
    - 犠牲者の選択（優先度、どこまで処理したか、影響度）
    - ロールバック（どこまで戻すか、一時退避、チェックポイント）
    - 飢餓状態（犠牲者の偏り、経費要因とロールバック回数）

デッドロック処理技法（まとめ）

名称	考え方	処理方法	処理負荷	利用効率	利便性
防止（静的）	循環待機の条件成立を防止	資源型に順序番号。各プロセスは、若番の資源型から順に要求。	○	×(注2)	×(注4)
回避（動的）	資源割り当てを制限しﾃﾞｯﾄﾞﾛｯｸの可能性を回避	資源要求の度に、銀行家ｱﾙｺﾞﾘｽﾞﾑで安全な順序の存在をﾁｪｯｸ。存在しなければ資源を割当てない	×(注1)	△(注3)	△(注5)
検出と回復	資源割り当てを制限せず、ﾃﾞｯﾄﾞﾛｯｸ発生を許容	検出ｱﾙｺﾞﾘｽﾞﾑを定期的に実行 ﾃﾞｯﾄﾞﾛｯｸのﾌﾟﾛｾｽをﾛｰﾙﾊﾞｯｸ or 資源の横取り	△	○	○
放置	何もしない（ユーザまかせ）	固まったらﾏﾆｭｱﾙで再開（Unix、Windows含むいくつかのOS）	○	○	×

注1：ﾌﾟﾛｾｽが資源を要求する度に銀行家ｱﾙｺﾞﾘｽﾞﾑを実行する必要がある
注2：使用しない資源を長時間確保することがある（すぐ使わなくても要求が必要）
注3：ﾃﾞｯﾄﾞﾛｯｸになるとは限らない場合でも、資源の割り当てを制限する
注4：資源の利用順序を守るようにﾌﾟﾛｸﾞﾗﾐﾝｸﾞする必要がある
注5：使用する資源の最大数を宣言する必要があるが、予測が困難な場合がある