デッドロック

- 一部の資源は同時に一つのプロセスしか利用できない
  →例）プリンタ：複数のプロセスを同時に受け入れる出力は・・・
- うまく処理するためには、
  - スプーリングデーモンのように直接操作プロセスを１つとする
  - 資源利用中はそのプロセスに占有させ、他のプロセスのアクセスをブロックする（相互排他）
- 相互排他の操作は資源の確保（ロック）と開放→この間に資源が利用される
- 複数のプロセスが２つ以上の資源を必要とし、お互いが取り合って処理が進まない状態をデッドロックと呼ぶ

横取り不可能な資源

- 資源には、横取り可能（プリエンプティブ）なものと横取り不可能（ノンプリエンプティブ）なものがある
- ノンプリエンプティブな資源は処理を中断して他の処理を実行できない（もしくはできても結果が破綻する）
- 占有が必要な資源に対してプロセスは、利用要求→利用→開放の手順を踏む
- 要求した時許可が出ない（他のプロセスが利用中）場合、要求したプロセスは開放されるまで待つ
  →プロセスは待ち状態

資源デッドロック

- あるプロセス集合の全てが自プロセス以外のプロセスが発するイベントを待ち続けている状態→デッドロック
- イベントが資源の開放である場合
  →資源デッドロック
- 発生条件（発生する可能性がある）
  1. 相互排他
  2. 確保待ち
  3. 横取り不可能
  4. 循環待ち

デッドロックへの対処

- 検討すべきは発生頻度と作業へのダメージ（コスト）
- 発生頻度が低く、作業へのダメージも少ない
  →ダチョウのアルゴリズム
- 予防策をわざと取らない（無視をする）
  - つまりなかったことにするのが簡単
    →再処理やプロセスリセット、システム再起動
- その他の場合
  - 回復：デッドロックが起きる前の状態にデータを復元する
  - 回避：デッドロックが起きそうな場合、資源の割り当てをやめる
  - 防止：デッドロックが起きなような機構を用意する

デッドロックへの対処

デッドロックの検出

- 循環待ちとなる関係を見つけ出す→資源割り付けグラフを使って状態を表現
- 循環待ちをプログラムで検出する

デッドロックからの回復

- 基本は循環待ちの解消
- プロセスの強制終了：影響が異なるので上手に選択を
  - 循環待ちを生じさせているプロセス
  - 循環待ちではないが、待ちの要因となる資源を確保しているプロセス
- ロールバック：実行するタイミングをずらす
  - 循環待ちになる前の時点まで、プロセスの状態を巻き戻す
  - 巻き戻す時点を定期的に記録する必要がある→チェックポイント
    - チェックポイントの記録にはコストがかかる→記録頻度を増やせば、ダメージは減らせるが・・・

デッドロックの回避

- デッドロックへのもう一つの対処方法は回避の仕組み
- 資源の割り当てを慎重に行う→デッドロック発生可能性を予測して判断する
- お金の貸付をモデルにした危険予測→銀行家のアルゴリズム
- 顧客（プロセス）がお金の借入（資源確保の要求）をした場合、銀行（OS）が資源の貸付（資源割当）をしても大丈夫かどうかを判断する
- プロセス全体の資源の貸借表を作成して、安全な状態かを調べる

割付表と状態の判断

- 割付表を安全な状態に保つ→デッドロックが発生しない
- プロセスの最大要求に答える＝プロセス終了＝資源開放を進めて全ての処理が終えられるか→終えられる＝安全な状態、終えられない＝安全でない状態
- 割り付け要求を認めた場合、安全でない状態になるなら割り当てを延期

複数種類の資源の場合

- 資源割付の表を拡張する→ベクトルと行列
  - プロセス名：P1, P2, … ,Pn
  - 各資源の数→*E*：存在資源ベクタ
  - 割り付け可能な資源の数→*A*：割り付け可能資源ベクタ
  - 割り付け資源の数→*C*：割り付け行列
  - 各資源の数→*R*：要求行列

- 判定のアルゴリズム

1. Rのマークのない行でA以下のものを探す
2. 有る場合、その行にマークを付け、Cで対応する行の値をAに加え、 1 に戻る
3. 無い場合、マークが全ての行に付いていれば安全、一つでも残っていれば安全でない

判定は難しい

- 銀行家のアルゴリズムにおいて安全で無い状態＝デッドロック状態では無い→デッドロックする可能性があるというだけ
- やりすぎるといつまでも割り付けしてもらえないプロセスが発生する場合も→スタベーション（飢餓状態）で処理が進まない
- 単純な銀行家のアルゴリズムの場合、
  - 資源要求の最大数が情報として必要
  - プロセスの数、資源の数が一定で変化しないのが想定

があるが、どちらも容易では無いので工夫がいる

デッドロックの防止

- デッドロックの対処法の一つで、条件が発生しないような機構を用いる
- 相互排他条件への対処→資源の仮想化
  - 資源への処理を非同期化して資源確保時間を減らす
  - 非同期のための処理がアダになる場合も
- 確保待ち条件への対処→資源の事前確保
  - プロセス実行開始時点で確保し、確保できたら実行する
  - ２相ロック（資源の完全確保→処理実行）という方法もある
  - 確保時間が長くなり作業効率は下がり、再作業（やり直し）のコストは上昇
- 横取り不可能条件への対処→横取り可能な仕組みへ変更
  - 相互排他と同様に仮想化、非同期化によって横取り可能な資源へと転換
  - 処理順が重要な資源には対応できない
- 循環待ち条件への対処→資源要求の制限
  - 単純な方法は、複数資源の確保を禁止
  - 複数資源を許容する場合、要求する資源の順番を全プロセスで統一する
  - 要求順がA→Bは良いが、B確保の後A要求はダメなら循環しない
- いずれも適用範囲に限度があり、組み合わせて使用

関連する諸問題

- デッドロックの完全な対策は難しい→状況にあった対処が必要
- 通信デッドロック：分散システムなどにおいて、メッセージ交換中のデータロスなどにより、双方が待ち状態になる
  - 対処はタイムアウト→通信のやり直し
- ライブロック：並列なプロセスが同時に処理を進め、途中で資源不足に陥り、処理の巻き戻し、再開が繰り返される
  - 対処は乱数による再開タイミングの拡散
- スタベーション：資源の必要数が多く、必要な資源が揃う前に他のプロセスの要求があり、永久的に割り付けられない状態が続くこと
  - 実行条件が満たせず状況が進行しないという現象自体はデッドロック、ライブロックと同じ