プロセス間通信

- プロセスはプロセッサもメモリも全て独立
  →他のプロセスがどのような状況でも動作を保証
- 作業の効率化には情報共有した上での並列作業
  →他のプロセスの作業情報なしでは同じ作業を繰り返す
- 他のプロセスと状況を教えあう仕組み
  →プロセス間通信

  IPC (Inter Process Communication)
- シグナル
- プロセスにイベント通知を行う方法
- シグナルを受け取ったプロセスは処理をやめ、シグナル名で決められた処理（シグナルハンドラ）を実行する
- 共有メモリ
- プロセス間でデータ共有をするための基本的な方法
- 互いのプロセスが持つメモリ空間の一部を同じ物理メモリで共用
- 一方で書き込んだ内容は、もう一方にも反映される
- パイプ
- プロセスの出力と入力を繋ぐための仕組み
- 一方のプロセスの出力結果をもう一方の入力データとして利用できる
- プロセスの間にFIFOのバッファ作られる
- ソケット
- 同一マシン上で動作していないプロセス間での通信に使用（同一マシン内でも可）
- ネットワークを介して、それぞれのホストが持つバッファの読み書きを行う
- 信頼性の高いTCPソケットや遅延の少ないUDPソケットがある

競合状態と相互排他

- 複数のプロセスが共有するデータには注意が必要
  - 同じデータへの書き込みが同じタイミングで発生
  - 書き込んでいる途中で読み込みが進行
  - 読み込んだデータを処理して書き込む途中で他者がデータ更新
- →競合状態

- 相互排他
- 競合状態をなくすには
- トラブルになりそうな場合は、対象の作業が終わるまで他のプロセスに干渉させない
- 資源を一つのプロセスに占有させるための制御
  →相互排他または排他制御
- 競合状態が起きないような仕組み

きわどい領域

- クリティカルリージョン、クリティカルセクション
- 他のプロセスが干渉すると問題になりそうな処理のこと
- この領域の処理に競合が発生しなければ，システムは健全に動作
  →いくつかの条件を保つことが必要
- きわどい領域の処理が常に一プロセスに限定されれば問題ない→条件１
- きわどい領域の処理を複数のプロセスが実行しようとした場合
  →一つを残して制止（ブロック）することで実現
- ブロックされたプロセスは終わるまで待機→終わったら別の１つが実行
- 健全な制御が行われるためには
- 必要ないところで他のプロセスをブロックしない→条件２
- プロセスによってえこひいきしない→条件３
- いつまで待っても順番が来ないのはダメ→条件４

フラグを使った相互排他ビジーウェイト

- 共用のフラグによって、きわどい領域が実行されているかを確認する方法（トイレの鍵）
- 他のプロセスはフラグ確認のループを回して待つ
  →ビジーウェイト
- 問題：鍵が開いていることを確認して、ドアを開けた時に他人が一緒に入ってしまうこと
- ドアを確認した時には、他者からはロック状態に見えてほしい
  →ハードウェアによる援助（TSL命令：test and set lock）
- 読み取りと書き換えの１命令化→アトミックオペレーション
- ビジーウェイトの問題点
- きわどい領域への待ちプロセスはチェック–ウェイトの繰り返し→プロセッサが常に稼働
- 待つプロセスは待ち状態にして、必要時に起こしてもらう仕組み→セマフォ

セマフォ

- セマフォのフラグsの操作を統一的に行う
  →関数として提供
- down操作（P操作）：
  - きわどい領域に入れるかチェック
  - s≧1ならsを一つ減らして領域内の処理に進む、s=0なら待つ
- up操作（V操作）：
  - sを一つ増やす
  - 待っているプロセスがあるかチェック
  - 待ちプロセスがあれば、起こす

- バイナリセマフォ
- ビジーウェイトの代替となるプロセスの相互排他
- sが0なら満室、sが1なら空き部屋ありで1プロセスなら入室可
- セマフォの問題点
- 独立した二つの操作（関数） down, upをきちんと組み合わせて使用しなければならない
- downはしたが、upしなかった→密室化
- プログラマのコーディングミスによって発生
  →システムでチェックできない
- システムに組み込んだ排他制御→モニタ

カウンティングセマフォとモニタ

- カウンティングセマフォ
- セマフォには別の使い方がある
- 複数のプロセスが同時に利用できるきわどい領域
- 例：公衆トイレ（個室が複数）
- セマフォsを1より大きい数値に設定
- s=5ならdownは5回まで可能→同時に5つのプロセスがきわどい領域を実行

- モニタ
- 特別定義の関数、変数、データ構造によってシステム側で排他制御
- プログラマは排他制御したい関数に特定の属性を指定→コーディングミスによる手続きの欠損はなくなる
- Java synchronized→指定した関数は同時に１つしか呼び出されない（後から呼び出されたら、前の処理が終わるまで待つ）

基本的な複数プロセスの問題

- 実際に複数プロセスを用いた処理を行う場合に発生しうる典型的な競合問題
  - プロデューサ–コンシューマ問題
  - リーダ–ライタ問題
- セマフォを用いることで問題を解決できる

- プロデューサ–コンシューマ問題
- 古典的な複数プロセスの同期問題
  - 生産者（プロデューサ）：キューにデータを追加
  - 消費者（コンシューマ）：キューからデータを取出し
- 条件
  - キューが満杯の場合、生産者は空きが出るまで待つ
  - キューが空の場合、消費者は追加されるまで待つ
- セマフォ
  - mutex →バイナリセマフォ
  - fill, empty →カウンティングセマフォ

- リーダ–ライタ問題
- 古典的な複数プロセスの並列化問題
  - リーダ：読み込みプロセス
  - ライタ：書き込みプロセス
- 条件
  - それぞれが対象データに所定の動作を行うにはアクセス許可が必要
  - リーダとライタにアクセス許可を同時には出せない
  - 複数のリーダには同時に許可しても良いが、ライタは一つのみ
- セマフォ
  - data →バイナリセマフォ
  - mutex →バイナリセマフォ