第8章 多重プロセス

花田 英輔
（このPowerPointは渡辺名誉教授作成のものを花田が一部改編した）

1

多重プロセス、多重スレッド(教科書8.1)

- 複数のプロセス/スレッドで構成されたプログラム
  - 1つのプログラムの中で複数の制御(実行順序)の制御が必要な場合のお話
- 複数のプロセスに分担させると便利な例
  - ネットワークからの入力を待ちながら、主たる処理をしたい
  - 利用者からの入力を待ちながら、主たる処理をしたい
  - 同時に複数の端末にサービスするプログラムを作りたい
  - 同時に複数の入出力装置を利用するプログラムを作りたい
    - 以下、「プロセス」で説明するが「スレッド」でも同様

2

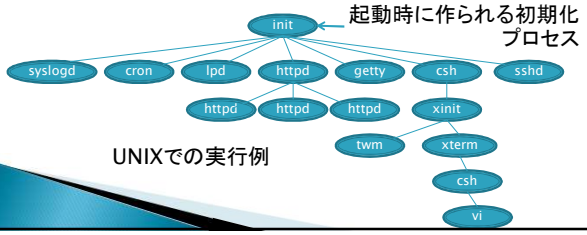
多重プロセス、多重スレッド(教科書8.1)

- 複数のプロセスが分担する場合の注意点
  - 各プロセス/スレッドの処理は独立させた方が良いが完全には独立できない場合あり
    - 互いに連絡を必要とする場合
    - 互いの実行順序の調整が必要な場合
- プロセス/スレッド間で同期を取る必要性あり
  - プロセス同士で同期実現は面倒⇒OSに同期処理の機能

3

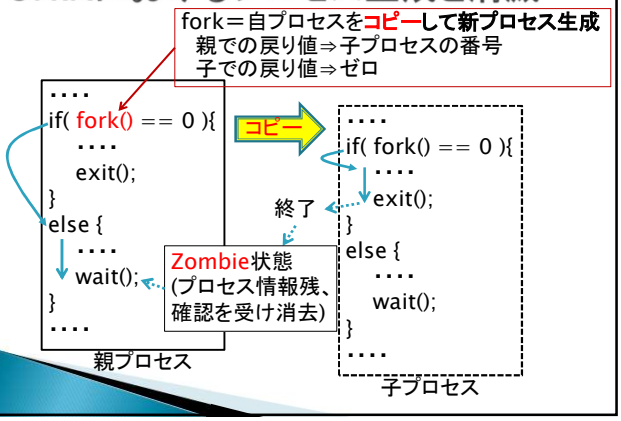
プロセスの生成と消滅(教科書8.2)

- 生成：プロセスがプロセスを生成
  - 作る方＝親プロセス、作られる方＝子プロセス
- 消滅：親が停止、または自分で停止
  - 子の生成から消滅までを、親が世話
  - 親が先に死んだら、トッププロセス（init）が世話



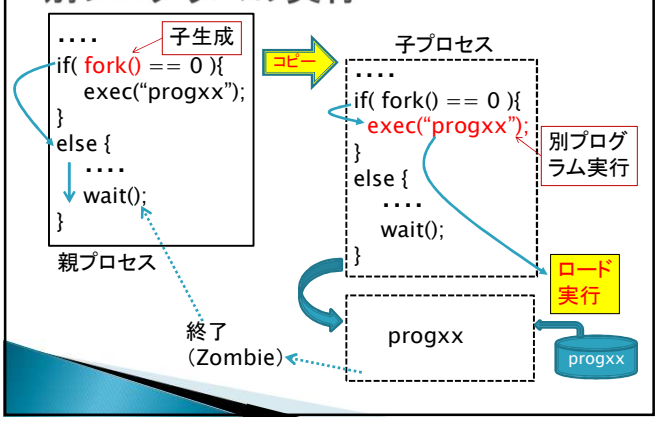
4

UNIXにおけるプロセス生成と消滅



5

別プログラムの実行



6

プロセス間の同期機能(教科書8.3)

- 並列で処理をしているとき、相手の処理状況を把握して進める必要性がある
  - 相手が作業している途中のデータでは？
  - 相手が事前に必要な処理を終えているか？
  - 処理が終わったら結果を渡して！
  - 勝手に進めるな！
  - ・・

7

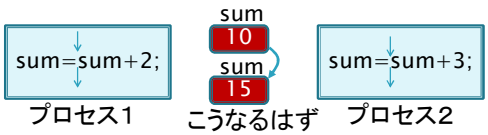
プロセス間の同期機能とOS

- OSがサービスする同期機能
  - 排他制御機能：重要な処理中に他が邪魔しないよう制御
  - 事象の連絡機能：ある事象が発生したことを連絡
  - プロセス間通信機能：プロセス間でのデータの受け渡し

8

排他制御の必要性

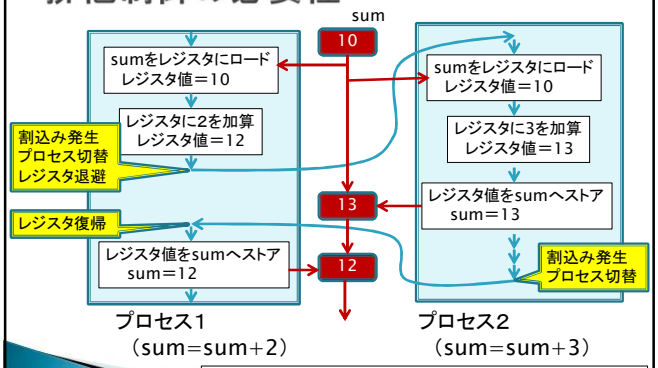
- 共用変数sumの値を2つのプロセスが扱う



- プロセスは切り替わりながら実行
  ⇒もしも計算途中で切り替わると？

9

排他制御の必要性

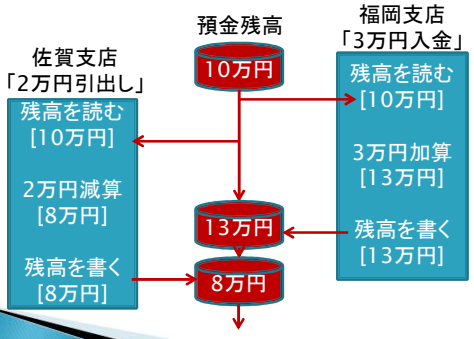


途中で切り替わると最終結果がおかしい

10

排他制御の必要性

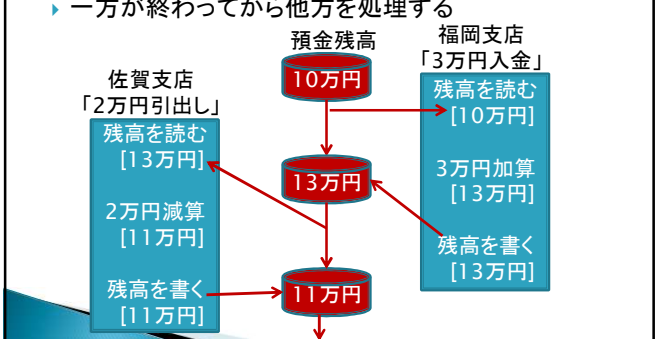
- 分かりやすい例示：銀行預金残高



11

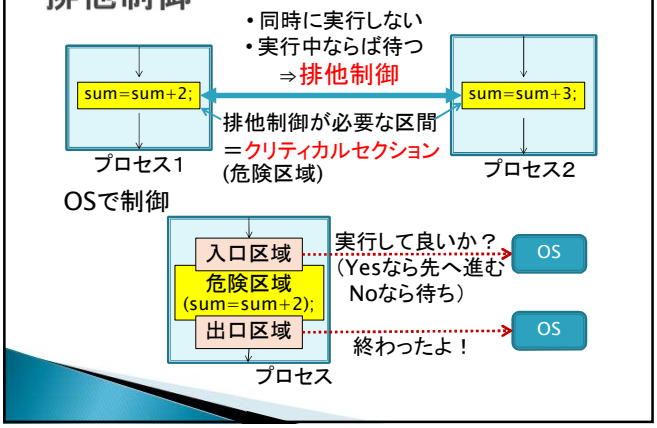
排他制御の必要性

- 間違いを防ぐには同時に処理しない（＝排他）
- 一方が終わってから他方を処理する



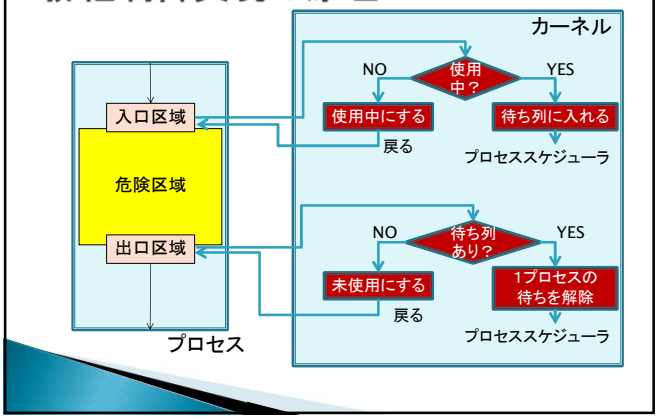
12

排他制御



13

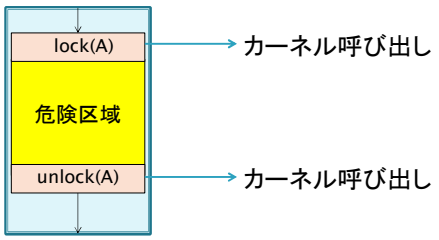
排他制御実現の原理



14

lock/unlockによる排他制御

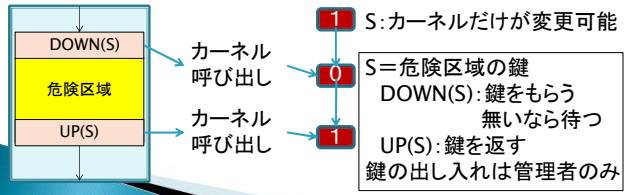
- ロック変数：下記ルーチンでのみアクセス可能
  - lock(A)：ロックAを確保する。確保できなければ待つ。
  - unlock(A)： ロックAを解放する。



15

セマフォを用いた排他制御

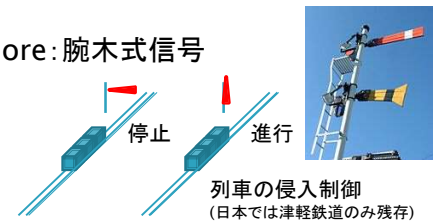
- セマフォ：下記ルーチンのみでアクセス可の整数変数
  - DOWN(S) ： セマフォSの値をdecrement(−1)する。
    結果が非負ならば、次の処理へ進む。
    負ならば、プロセスは待ち列へ。
  - UP(S) ： セマフォSの値をincrement(+1)する。
    結果が正ならば次の処理へ進む。
    非正ならば、プロセスを待ち列から選択、レディへ。



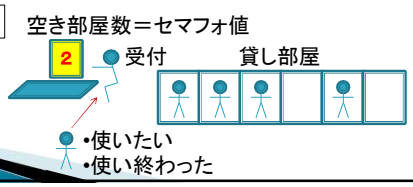
16

セマフォ

- semaphore：腕木式信号



分かりやすい例



17

モニタ

- Lock/Unlock、Up/Downでは開放忘れの可能性

```
lock(a);
・・・
if(・・・)return;
・・・
unlock(a);
```

- monitor＝排他実行機能を持つクラス
  - メソッドが排他実行(一つが実行中なら他は待たされる)
  - クリティカルセクションの素直な表現

```
monitor class Account {
  ・・・
  public method boolean withdraw(int amount)
  {  ・・・・ }          引き出す
  public method deposit(int amount)
  {  ・・・・ }          預ける
}
```

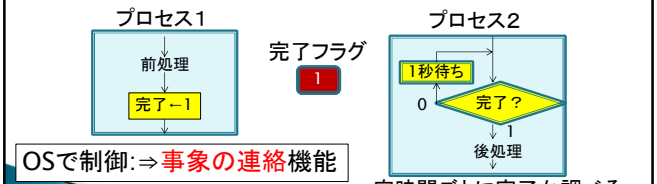
18

事象の連絡(教科書8.4)

- どちらが先でも良いから一つずつ
  - OSで制御⇒排他制御機能

例)あるプロセスの処理が他プロセス処理に依存(実行順固定)

- データ収集が終わったよ、データ解析開始して！
- 応用プロセス同士では面倒・非効率(下図)

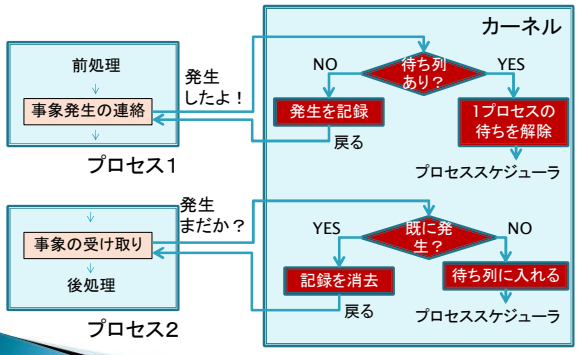


OSで制御:⇒事象の連絡機能

一定時間ごとに完了か調べる(ポーリング)⇒効率悪い

19

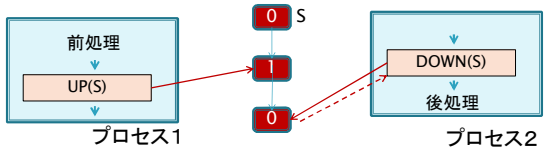
事象の連絡機能実現の原理



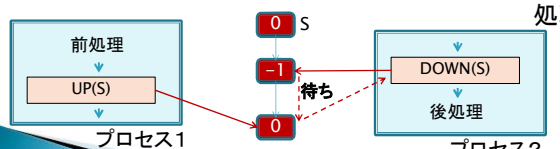
20

セマフォを用いた事象の連絡

プロセス1が先に到着⇒プロセス2は待ちなしで次の処理へ



プロセス2が先に到着⇒プロセス1の到着を待ってから次の処理へ



セマフォ初期値:排他⇒1、連絡⇒0

21

Javaの同期機能

synchronized＝排他実行を指示

生産者スレッド

```
// dataの生産
・・・・・
synchronized(buffer){
   while(bufferFill==ture){
      ・・・
      buffer.wait();
      ・・・
   }
}
//バッファに入れる
buffer.put(data);
synchronized(buffer){
   bufferFill=true;
   buffer.notify();
}
```

消費者スレッド

```
synchronized(buffer){
   while(bufferFill==false){
      ・・・
      buffer.wait();
      ・・・
   }
}
```

連絡待ち

```
//バッファから取り出す
data=buffer.get();
synchronized(buffer){
   bufferFill=false;
   buffer.notify();
}
```

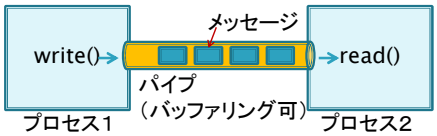
連絡発行

```
// データの消費
・・・・
```

22

プロセス間の通信(教科書8.5)

- プロセス間通信機能(IPC、Inter Process Communication)
  - プロセス間でデータ(メッセージともいう)を送受
  - 排他・同期より高機能(多量の情報を交換できる)
- UNIXにおけるIPC機能⇒パイプ

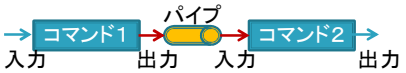


シェルのコマンドとしても指定可能(｜:縦棒記号)
aa｜bb
⇒プログラムaaの標準出力をプログラムbbの標準入力にパイプ接続

23

パイプ

- 2つのコマンドをつなぐ
  コマンド1 ｜ コマンド2 :コマンド1出力をコマンド2入力へ



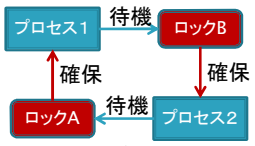
- 使用例

```
$ ls｜wc -l    ←ファイル一覧出力を、行の計数へ
10    ←ここにファイルは10個(lsは1行1ファイル表示のため)

$ ls｜grep “txt”   ←ファイル一覧出力を文字列探索へ
test.txt       ←ファイル名に”txt”を含むファイル一覧
aa.txt

$ ls｜grep “txt”｜wc -l   ←ファイル名に“txt”を含むファイルの個数
```

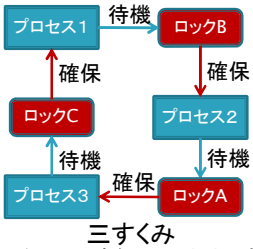
24

デッドロック(教科書8.6)

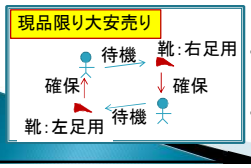
- 同期の取り方がまずくて処理が先に進めなくなること



単純なデッドロック



三すくみ



- ロックを一つ確保したまま、もう一つを待つと起こる
- ロック順を決めておけば、防げることが多い

25

今回の課題

1. セマフォを使うと排他制御ができる。2つのプロセスA,Bがクリティカルセクションの入り口に前後して到達したときのSの値の変化を示せ。
2. 「デッドロック」について説明せよ
3. (予習)メモリの「断片化」とは何か説明せよ

(1.のプロセスの実行形態等はMoodle内ファイルを用いること)

- 今回のファイル名は“学籍番号-OS10.docx”
  (例:23238000-OS10.docx)としてください
- 締切:12月13日(金) 18:00 (遅れた場合は減点)

26