コンピュータシステムII（９）

情報基盤センター
天野　浩文

1

演習問題の解答例　1.

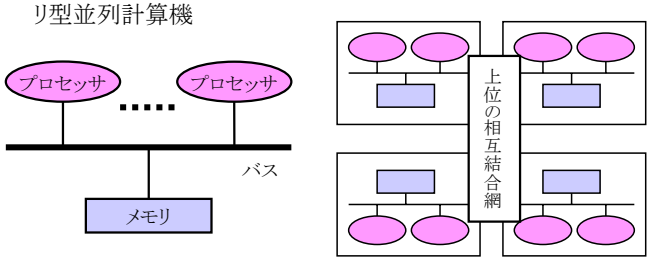
並列処理・分散処理では，逐次型処理では考慮する必要のなかった以下のような事項を考慮する必要があるため，逐次型処理よりも困難になる．

- 異なるプロセッサ（計算機）で進んでいく複数の処理の間で，必要なデータを交換しなければならないことがある．このためには，逐次型処理では必要のない通信処理を行ってデータ交換を行わなければならない．
- 複数の処理の間で，他のプロセッサ（計算機）での処理が一定の段階まで終了するのを待ち合わせなければならないことがある．このためには，逐次型処理では必要のない同期処理を行わなければならない．

2

演習問題の解答例　2.

- UMA型システム（左下）
  メモリのどの位置にアクセスする場合でも同じ時間で処理できるような共有メモリ型並列計算機
- NUMA型システム（右下）
  メモリの位置によってアクセス時間に差が生じるような共有メモリ型並列計算機



3

演習問題の解答例　3.

プロセス・スレッドのいずれも，並列処理や分散処理を行うときの処理のまとまりの一種であるが，以下のような違いがある．

- スレッドはプロセスよりも大きさが小さく，ひとつのプロセスが複数のスレッドから構成されることがある．
- プロセスは固有のメモリ空間を持つのに対して，同一プロセス内のスレッドはそのプロセスのメモリ空間を共有する．
- プロセス間の切り替えは，スレッド間の切り替えに比べて，処理に時間がかかる．
- 異なるプロセスどうしがメモリアクセスによって相互に影響を及ぼすことはないが，同一プロセス内のスレッドどうしは不適切なメモリアクセスを行うと相互に悪影響を及ぼすことがある．

4

演習問題の解答例 4.(a)

分散共有メモリ機構がない場合，リモートメモリへのアクセスは，通信処理を用いた処理依頼あるいはデータ転送依頼を含む，まったく別の処理として記述しなければならない．その一方，分散共有メモリ機構がある場合には，演算ノード内のローカルデータにアクセスする操作と，別の演算ノードのメモリ上にあるリモートデータにアクセスする操作を同じ記述形態で表すことが可能になる．

5

演習問題の解答例 4.(b)

- コンパイラ方式は，コンパイル時にリモートメモリへのアクセスを通信処理に書き換えて実行可能プログラムを生成する．このため，（コンパイル時にリモートメモリへのアクセスがローカルメモリへのアクセスと区別できるように）リモートメモリへのアクセスを表す特殊な構文を加えて，既存のプログラミング言語を拡張する．したがって，リモートメモリへのアクセスをある程度意識したプログラミングが必要で，新しいコンパイラも作成しなければならない．
- 実行時方式は，プログラム実行時に共有メモリへのアクセスが発生すると，その時点でOSに制御を移し，リモートメモリへのアクセスを代行させる．このため，言語の拡張や新しいコンパイラの作成は不要であるが，コンパイラ方式よりもやや複雑な機構が必要で，OSの機能拡張も必要である．

6

演習問題の解答例 4.(c)

(i)および(ii)は，いずれもコンパイラ方式で実現する必要がある．
一方，(iii)は，実行時方式で実現する必要がある．

7

演習問題の解答例 4.(d)

- データを複写することによって分散共有メモリを実現する場合には，リモートメモリの操作が必要になると，そのデータがローカルメモリにコピーされる．このとき，同じデータのコピーが複数箇所に存在する状態になる．これらのコピーに対する変更操作があると，同じデータの値が複数存在することになり，これを放置すると誤った計算結果を生成する可能性がある．このため，同一のデータのコピーに対する変更操作があった場合，それ以降の同じデータに対する参照操作が常に新しいデータにアクセスできるように保証することが必要となる．

8

演習問題の解答例 5.

- コピーを行わない方式
  - 長所:送受信が非常に速い.
  - 短所:プロセス間で共有できるメモリ空間が必要となる. 送受信するデータのアクセス保護が弱い.
- コピーを行う方式
  - 長所:送受信するデータのアクセス保護が強い.
  - 短所:送受信に時間がかかる.

9

演習についての注意事項

- 「解答例」に示したもの以外にも正解となる解答はありうる.
  - まだ採点を行っていないが, 採点時にその他の解答例が見つかれば, 次回以降の講義で補足説明を行う.
- 各設問の配点, 本科目の成績評価に占める割合は未定.
- 答案は返却しない.

10

排他制御機構と同期機構

11

処理全体の無矛盾性を考えなければならない例(1-1)

- 同じ銀行口座に入金や出金処理を行う場合

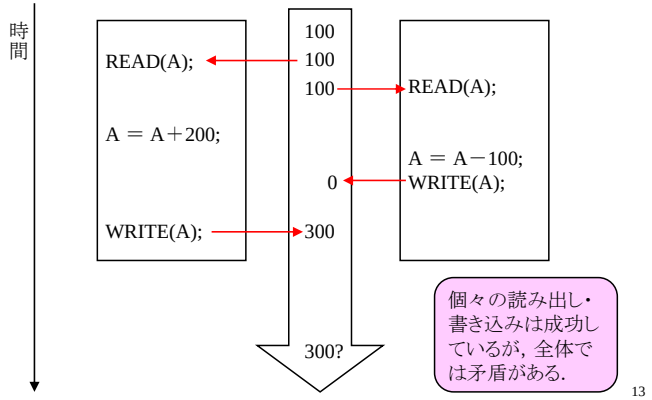
```
/* 入金処理 */

/* 口座の残高を調べる*/
READ(A);

/* 残高に入金額を加える */
A=A+200;

/* 口座の残高を書き戻す */
WRITE(A);
```

```
/* 出金処理 */

/* 口座の残高を調べる*/
READ(A);

/* 残高から出金額を引く */
A=A-100;

/* 口座の残高を書き戻す */
WRITE(A);
```

12

処理全体の無矛盾性を考えなければならない例(1-2)

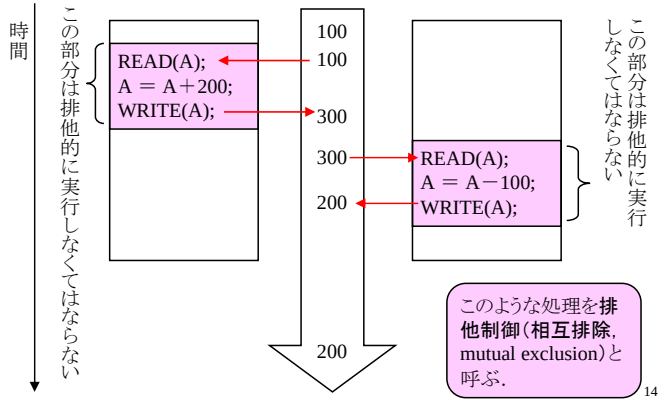
● これが複数の演算ノードで処理されると…



個々の読み出し・書き込みは成功しているが，全体では矛盾がある．

処理全体の無矛盾性を考えなければならない例(1-3)

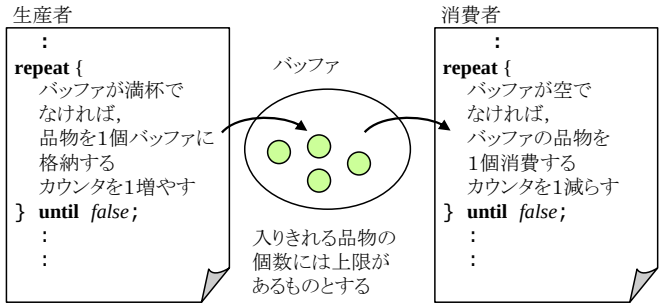
● 正しく処理するためには…



このような処理を**排他制御(相互排除**, mutual exclusion)と呼ぶ.

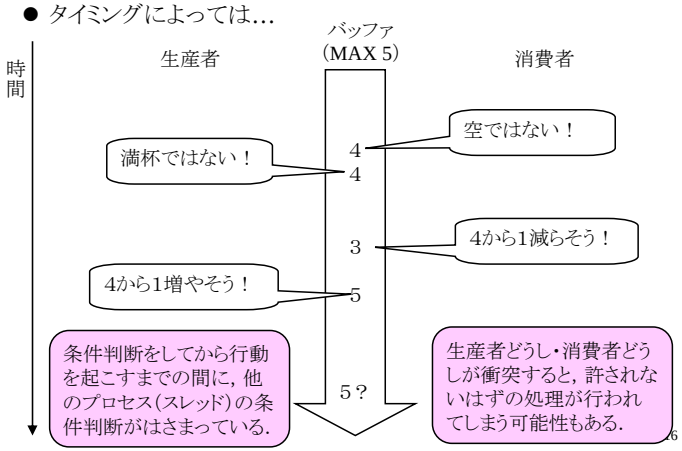
処理全体の無矛盾性を考えなければならない例(2-1)

● 有限バッファを用いた生産者・消費者問題
(bounded-buffer producer/consumer problem)



処理全体の無矛盾性を考えなければならない例(2-2)

● タイミングによっては…

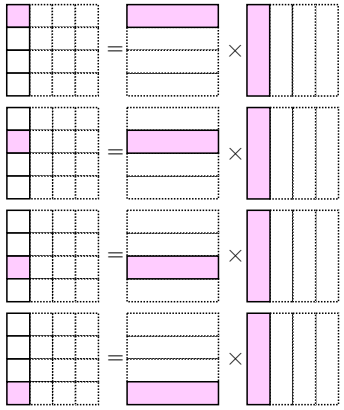


条件判断をしてから行動を起こすまでの間に，他のプロセス(スレッド)の条件判断がはさまっている.

生産者どうし・消費者どうしが衝突すると，許されないはずの処理が行われてしまう可能性もある.

処理全体の無矛盾性を考えなければならない例(3-1)

- 行列積 C=A×B を4プロセッサで計算する場合
  - プロセッサ1は，自分の受け持ち部分を計算する．
  - 計算が終わったら，行列Aの次の行ベクトルを隣のプロセッサから受け取り，次の担当部分を計算する．
  - その計算が終わったら，また，行列Aの次の行ベクトルを隣のプロセッサから受け取り，次の担当部分を計算する．
  - 受け持ち部分が最終行まで来れば，終了．
- 同じような処理を，他のプロセッサでも行う．

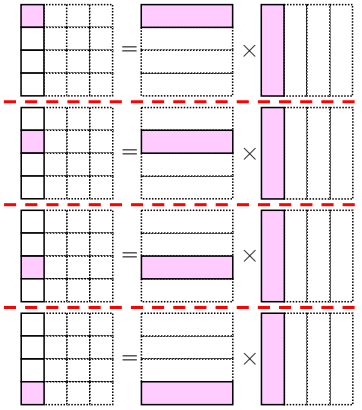


17

処理全体の無矛盾性を考えなければならない例(3-2)

- 第一段の計算が全プロセッサで終了する前に行列Aの行ベクトルの交換が始まってしまうと…
  - 計算の遅れているプロセッサでは，受け持ち部分の計算が正しく終了できなくなる．
- したがって，右の図の赤い破線のところで，**他のプロセッサでの計算の終了を待ち合わせる**必要が出てくる．

このような処理をバリア同期(barrier synchronization)と呼ぶ．



18

排他制御と同期処理

- 排他制御(相互排除)
  - 複数のプロセス(スレッド)のプログラムコードの中に，共通の資源(例:共有変数など)にアクセスする部分があるとき，その部分を**同時に実行できるプロセス(スレッド)がただ一つとなるようにする**こと．
    - 最初のプロセス(スレッド)が上記のような**危険区間**(critical section)に到達すると，他のプロセス(スレッド)は危険区間の中に入れないように制御する．
- 同期処理
  - プロセス(スレッド)が，**何らかの条件が成り立つまで処理を中断して待つ**こと．
    - ここでいう「条件」とは，自分自身以外のものによって変更される種類のものを指す．

19

セマフォ(semaphore)

- もともとは相互排除のための機構
  - 少し工夫すると，ある種の同期にも使える．
- 最も簡単な相互排除機構
- 以下のような性質を持つ整数型変数
  - 初期化を別にすれば，次のような P および V と呼ばれる2種類の操作によってのみアクセス可能
    - P: 危険区間の開始を宣言する．**他のプロセス(スレッド)が危険区間に入っているときには，先に進まずにここで待つ**．
    - V: 危険区間の終了を宣言する．**同じセマフォに対するP命令で待っている他のプロセス(スレッド)が先に進めるようになる**．

20

セマフォによる相互排除の例(1)

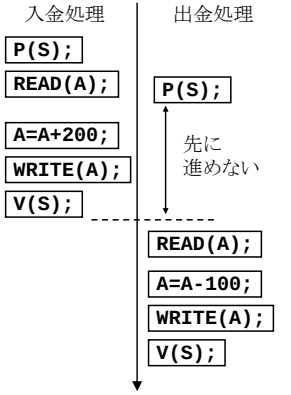
```
/* 入金処理 */
/* 危険区間の開始 */
P(S);
READ(A);
A=A+200;
WRITE(A);
V(S);
/* 危険区間の終了 */
```

```
/* 出金処理 */
/* 危険区間の開始 */
P(S);
READ(A);
A=A-100;
WRITE(A);
V(S);
/* 危険区間の終了 */
```

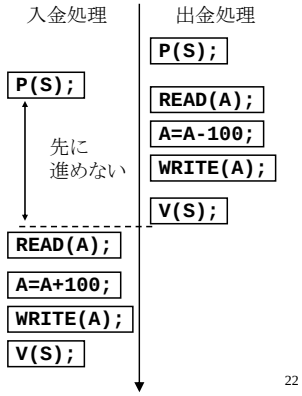
21

セマフォによる相互排除の例(2)

- 入金処理のほうが先に**P(S)**を実行したとき



- 出金処理のほうが先に**P(S)**を実行したとき



22

PV操作の実現法の例(1)

```
P(S): while S<=0 do {};
        Sを1減らす
```

```
V(S): Sを1増やす
```

- S≦0 のときは危険区間に入れない．S＞0 のときは危険区間に入れる．
  - 初期値を1にすると…
    - 危険区間に同時に入れるプロセス(スレッド)数は1．
    - このようなセマフォは，0 (入れない)または 1 (入れる)のいずれかの値しか取らない．
      ⇒**バイナリセマフォ**
  - 初期値を1より大きくすると，危険区間に同時に入れる数が1より大きくなる．
- 危険区間に入れない間は，ずっと**空回り**を続ける．

23

PV操作の実現法の例(2)

- セマフォに待ち行列を付随させる方法もある．
  - 危険区間に入れないときには，自分自身を待ち行列につないで，休眠状態に入る．
  - 危険区間を出るときには，待ち行列の中からプロセス(スレッド)を1つ選んで，休眠状態から覚醒させる．

```
P(S): if S<=0 then {
        自分自身を待ち行列に登録する
        自分自身を休眠状態にする
      }
      Sを1減らす;
```

教科書 p.62 のセマフォはこの形式

```
V(S): if S<=0 then {
        Sを1増やす
        待ち行列のプロセス(スレッド)を1つ選び覚醒させる
      }
```

PV操作の実現法の例(3)

- セマフォに計数機能を持たせる方法もある.
  - セマフォの値は, 0, 1だけでなく, 任意の整数値を取りうる.
    - 初期値2以上のときは危険区間に同時に入れる数が1より多い.
    - 負の値になったときには, その絶対値が, 待っているプロセス(スレッド)数を表す.

```
P(S): Sを1減らす
      if S<0 then {
         自分自身を待ち行列に登録する
         自分自身を休眠状態にする
      }
```

値の検査よりも先に, Sの値を増減させていることに注意.

```
V(S): Sを1増やす
      if S<=0 then {
         待ち行列のプロセス(スレッド)を1つ選び覚醒させる
      }
```

25

セマフォによる同期処理の例(1)

- 「処理1が同期ポイントを通過した」という条件が成り立つまで, 処理2が同期ポイントで待つようにする
  - バイナリセマフォの初期値を「危険区間に入れない状態(=0)」にセットしておく.

処理1

```
  :
  :
  :
/* 同期ポイント */
V(S);
  :
  :
  :
```

処理2

```
  :
  :
P(S);
/* 同期ポイント */
  :
  :
  :
  :
```

26

セマフォによる同期処理の例(2a)

- 「処理1, 処理2が同期ポイントを通過した」という条件が成り立つまで, 処理3が同期ポイントで待つようにする.
  - セマフォの初期値を −1 にセットしておく.

処理1

```
  :
  :
  :
/* 同期ポイント */
V(S);
  :
  :
  :
```

処理2

```
  :
  :
  :
/* 同期ポイント */
V(S);
  :
  :
  :
```

処理3

```
  :
  :
P(S);
/* 同期ポイント */
  :
  :
  :
  :
```

27

セマフォによる同期処理の例(2b)

- 「処理1, 処理2が同期ポイントを通過した」という条件が成り立つまで, 処理3が同期ポイントで待つようにする.
  - バイナリセマフォを2つ用意し, 初期値を0にセットしておく.

処理1

```
  :
  :
  :
/* 同期ポイント */
V(S1);
  :
  :
  :
```

処理2

```
  :
  :
  :
/* 同期ポイント */
V(S2);
  :
  :
  :
```

処理3

```
  :
P(S2);
P(S1);
/* 同期ポイント */
  :
  :
  :
  :
```

この順序は入れ替わってもよい

28

セマフォの使い方のまとめ

- 排他制御に使うとき
  - 初期状態は，危険区間に入れる状態
    - セマフォの初期値を1または正の整数にセット
  - 各プロセス(スレッド)は，危険区間の前後をP操作・V操作ではさんでおく．
- 同期処理に使うとき
  - 初期状態は，P操作が先に進めない状態
    - セマフォの初期値は 0 または負の整数
  - 先行させたいほうにV操作を，後追いさせたいほうにP操作を実行させる．

29

バリア同期操作の実現法の概要

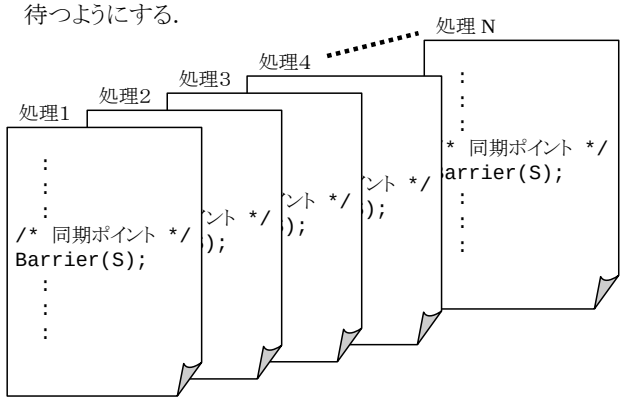
- 待ち状態のプロセス(スレッド)を1つだけ覚醒させるPV操作とは異なり，*N* 番目にそれを実行したものが他のすべてを覚醒させる．
  - バリア同期すべきプロセス(スレッド)の数 *N* を初期値とする共有変数を用いる

```
Barrier(S): Sを1減らす
            if S>0 then {
               自分自身を休眠状態にする
            } else {
               他のすべてのプロセス(スレッド)を覚醒させる
               Sを再初期化
            }
```

30

バリア同期の例

- すべての処理が同期ポイントに到達するまで，同期ポイントで待つようにする．



31

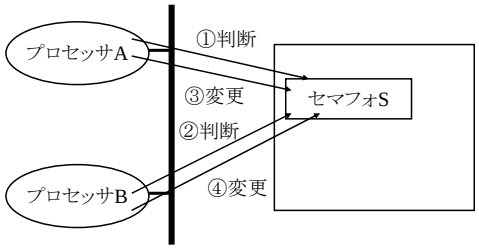
排他制御とシングルプロセッサ

- 排他制御は，単一のプロセッサ上で複数のプロセス(スレッド)が時分割方式で並行に動作している場合でも，必要になる．
  - スライドNo.12～No.16のような議論は，それぞれのプロセス(スレッド)がシングルプロセッサ上で時分割実行されたときにも，そのままあてはまる．
  - 実は，2年後期の「オペレーティングシステム」で勉強しているはず．
- ただし，並列処理・分散処理の場合には，**排他制御を実現するためのメカニズムはより複雑になる**．
  - シングルプロセッサ上では，プロセス(スレッド)の切り替えが起きないようにするだけでよかったが…
    - 割り込み禁止のマスクをかけるだけでも実現できる．

32

P操作の実装に関する注意

- 「先に進んでよいかどうか」の条件判断と，それに続くセマフォの変更や休眠処理までの間に，他のP操作の条件判断が行われないようにしなければならない．
  - この条件が守られないと，スライドNo.13で示したのと同様の不具合が発生する．
- 複数のプロセッサが共有メモリ上の変数にアクセスできる場合



Aで割り込みが禁止されていても，①②③④の順で実行されてしまう可能性がある．

33

ではどうするとよいか？

- ①と③の間に他の操作が行われないようにするには…
  - それが単一の機械語命令で実行されるようにすればよい．
    ⇒ハードウェア（CPU）にそのような命令が備わっている必要がある．
  - 言葉で言うのは簡単だが… もうひとひねり必要．
- `Test-and-Set` 命令を使ったP操作

```
while Test-and-Set(lock) do {}
```

- `Test-and-Set(lock)`
  - 変数`lock`の値を返す
  - `lock`には *true*を書き込む

`lock`の値が *true* なら，上のP操作は**TS命令の繰り返し**になる．
`lock`の値が *false* なら，上のP操作は何も実行せずに終了して**次に進む**．

34