並行処理の考え方

東邦大学

この節で学ぶのは
「並行処理」
で起こることがらです

プログラムが並行に動くと、
様子が大分変わります

東邦大学

はじめに「並行」とは

はじめに「並行」とは

- ポイントは「並列」vs「並行」

ポイントは「並列」vs「並行」

- 走っている人 (＝プログラム/プロセス) から見ると、どちらも同じ

ポイントは「並列」vs「並行」

- 走っている人 (＝プログラム/プロセス) から見ると、どちらも同じ
- 物理CPUから見ると

ポイントは「並列」vs「並行」

- 走っている人 (＝プログラム/プロセス) から見ると、どちらも同じ
- 物理CPUから見ると
  - 並列＝複数CPUで、本当に同時並列

ポイントは「並列」vs「並行」

- 走っている人 (＝プログラム/プロセス) から見ると、どちらも同じ
- 物理CPUから見ると
  - 並列＝複数CPUで、本当に同時並列
  - 並行＝CPUは１つで、プロセス切替による疑似的な（にせ物の）並列

ポイントは「並列」vs「並行」

- 走っている人 (＝プログラム/プロセス) から見ると、どちらも同じ
- 物理CPUから見ると
  - 並列＝複数CPUで、本当に同時並列
  - 並行＝CPUは１つで、プロセス切替による疑似的な（にせ物の）並列

プログラムやプロセスから見ると同じなので
しばらくは同じだと思っておきます

で、ここからは
プログラム（プロセス）から見た
並行（並列）の話です

まず、何で並行の話？

- 何で、OSの授業で並行（並列）の話か？

まず、何で並行の話？

- 何で、OSの授業で並行（並列）の話か？
- 前にも少し触れたが、たとえば用途として

まず、何で並行の話？

- 何で、OSの授業で並行（並列）の話か？
- 前にも少し触れたが、たとえば用途として
  - 例：　裏で印刷
  - 例：　複数のウィンドウ
  - 例：　サーバーで複数の要求を同時処理

まず、何で並行の話？

- 何で、OSの授業で並行（並列）の話か？
- 前にも少し触れたが、たとえば用途として
  - 例：　裏で印刷
  - 例：　複数のウィンドウ
  - 例：　サーバーで複数の要求を同時処理
- 更には、OS内部のサービスも、並行して動くプロセスとして作ると便利

まず、何で並行の話？

- 何で、OSの授業で並行（並列）の話か？
- 前にも少し触れたが、たとえば用途として
  - 例：　裏で印刷
  - 例：　複数のウィンドウ
  - 例：　サーバーで複数の要求を同時処理
- 更には、OS内部のサービスも、並行して動くプロセスとして作ると便利
  - Aのサービスをしている時にBを待たせない

まず、何で並行の話？

- 更には、OS内部のサービスも、並行して動くプロセスとして作ると便利
  - Aのサービスをしている時にBを待たせない
  - たとえば
    OSがプリンタを制御している時に

まず、何で並行の話？

- 更には、OS内部のサービスも、並行して動くプロセスとして作ると便利
  - Ａのサービスをしている時にＢを待たせない
  - たとえば

    OSがプリンタを制御している時に

    同時にネットワークが背景で動いていて

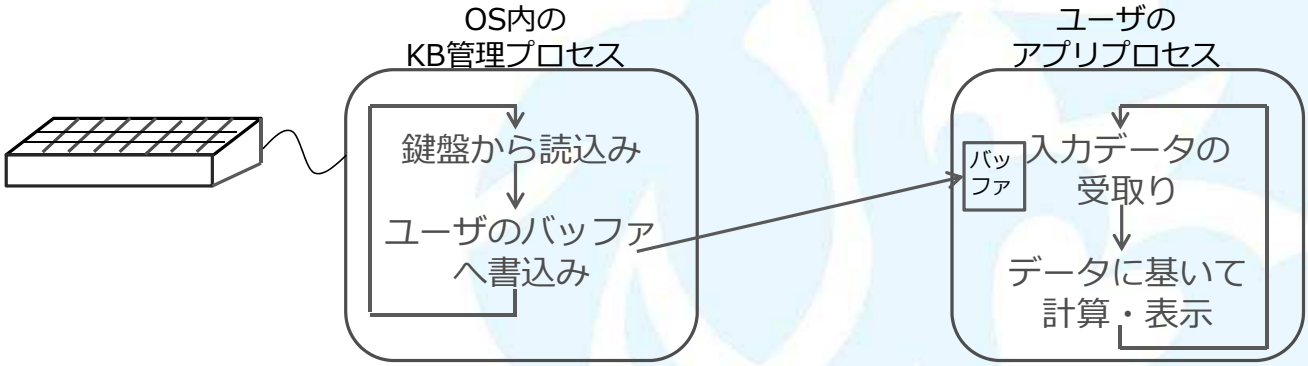
    更にマウス操作を受付けて…

戻って、プログラムから見ると？

- プログラムからどう見えるか？

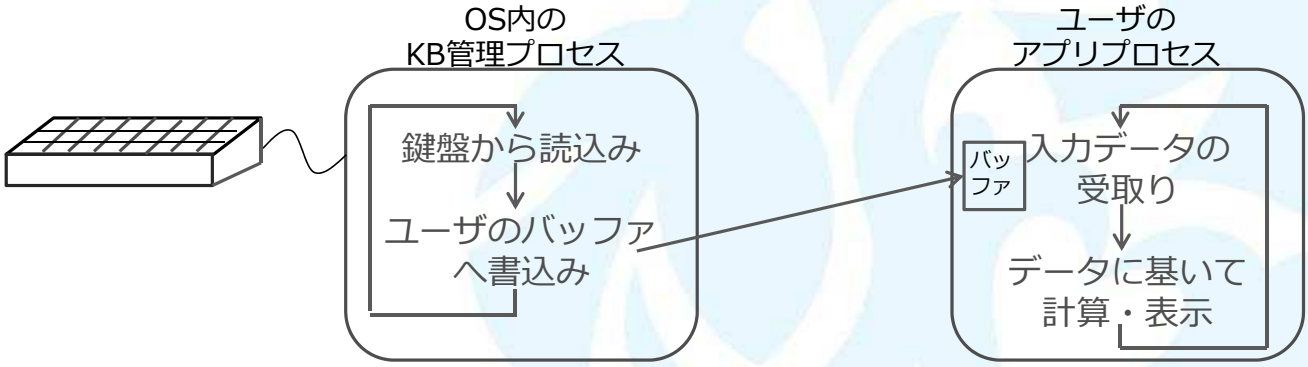
戻って、プログラムから見ると？

- プログラムからどう見えるか？
  - こんなイメージ：



戻って、プログラムから見ると？

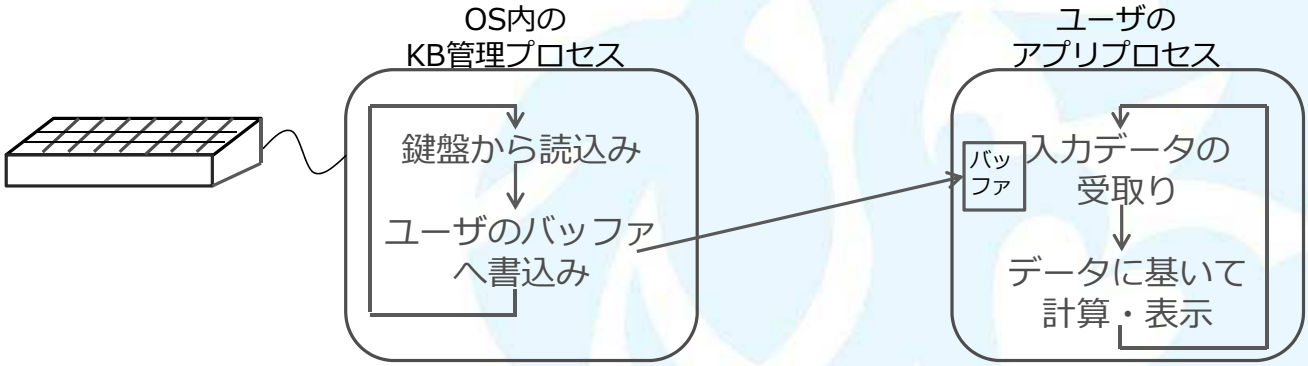
- プログラムからどう見えるか？
  - こんなイメージ：



OS内KB管理プロセスでは、KBから読んだデータを
バッファへ書き出してから、KBから次のデータを読む

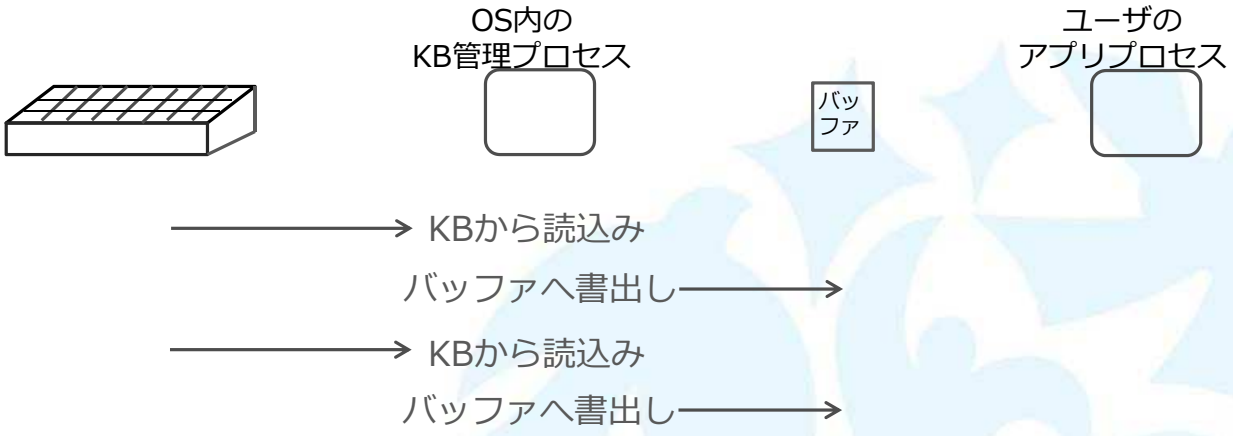
戻って、プログラムから見ると？

- プログラムからどう見えるか？
  - こんなイメージ：

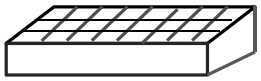


OS内KB管理プロセスでは、KBから読んだデータを
バッファへ書き出してから、KBから次のデータを読む
バッファへ書く前にKBから読むと前のデータが消え

戻って、プログラムから見ると？



戻って、プログラムから見ると？



OS内の
KB管理プロセス

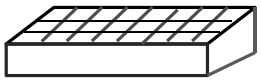
バッファ

ユーザの
アプリプロセス

→入力データの受取り
計算

→入力データの受取り
計算

戻って、プログラムから見ると？



OS内の
KB管理プロセス

バッファ

ユーザの
アプリプロセス

→ KBから読込み

バッファへ書出し→

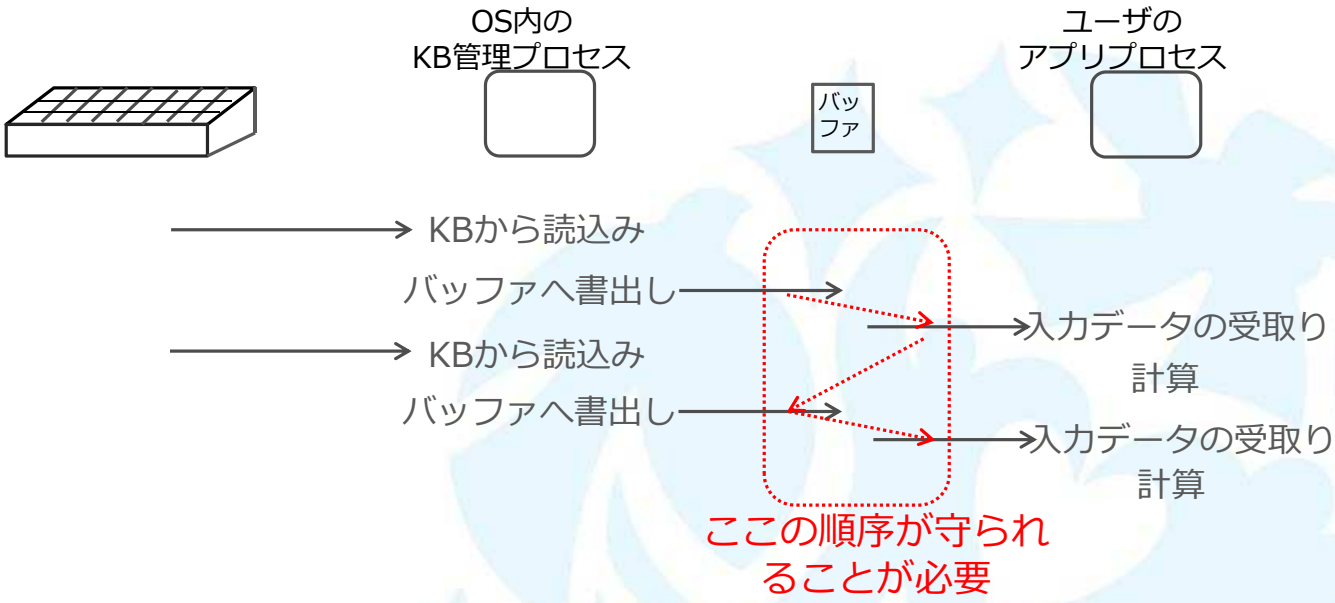
→入力データの受取り
計算

→ KBから読込み

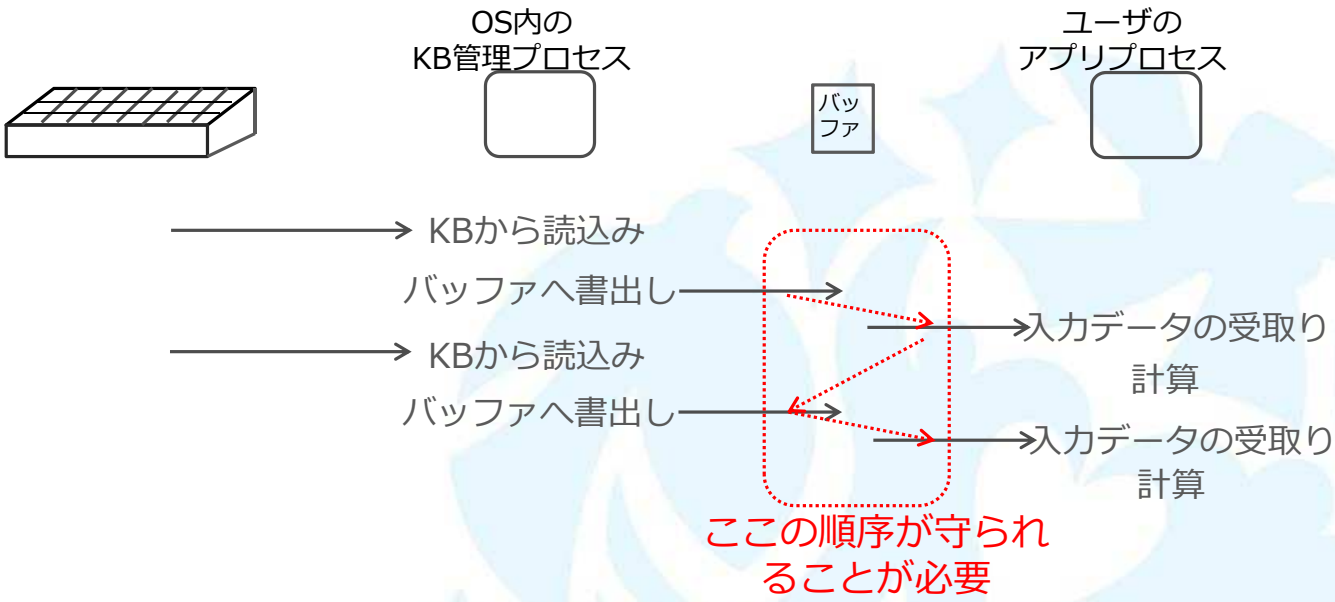
バッファへ書出し→

→入力データの受取り
計算

戻って、プログラムから見ると？

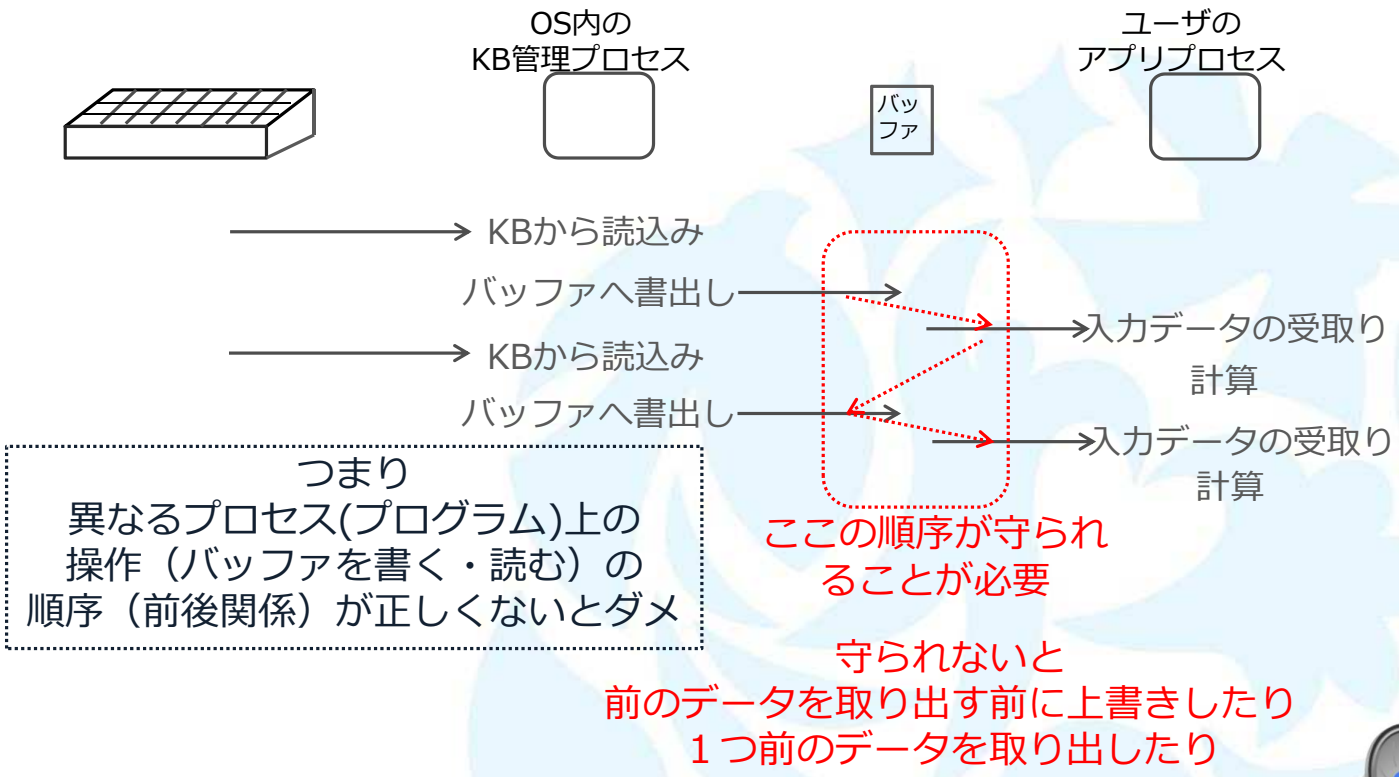


戻って、プログラムから見ると？



守られないと
前のデータを取り出す前に上書きしたり
１つ前のデータを取り出したり

戻って、プログラムから見ると？



戻って、プログラムから見ると？

- 異なるプロセス上での操作の順序が
  正しくないとダメ

ところが

戻って、プログラムから見ると？

- 異なるプロセス上での操作の順序が
  正しくないとダメ

ところが

- 異なるプロセス上での実行の状況 (位置) は
  お互いに分からない
  - つまり、どこを走っているか分からない、かつ
  - 他のプロセス内の変数の値も分からない

順序の制御のしようがない

戻って、プログラムから見ると？

それで

- 順序を制御したり　　同期

戻って、プログラムから見ると？

それで

- 順序を制御したり　　同期
- 同時にアクセスしないように　相互排他

戻って、プログラムから見ると？

それで

- 順序を制御したり　　同期
- 同時にアクセスしないように　相互排他

する仕掛けを用意する

ここまでのまとめ

- プロセス間の同期や相互排他は
  ２つのプロセスが共同して作業したい時
  [　　]
  道具である

ここまでのまとめ

- プロセス間の同期や排他制御は
  ２つのプロセスが共同して作業したい時
  [お互いのタイミングをとるための]
  道具である

ここまでのまとめ

- プロセス間の同期や相互排他は
  ２つのプロセスが共同して作業したい時
  お互いのタイミングをとるための
  道具である
- それが必要な理由は、並行に動くプロセスは
  [　　]
  [　　]
  分からないからである

や
が

ここまでのまとめ

- プロセス間の同期や排他制御は
  ２つのプロセスが共同して作業したい時
  お互いのタイミングをとるための
  道具である
- それが必要な理由は、並行に動くプロセスは
  お互い相手が今何を実行しているか、
  相手の変数の値がどうなっているか、
  分からないからである

や
が

並行処理の考え方が掴めましたか？

○

次へ

×