デバイス管理とは

この節で学ぶのは
ＯＳの中で
（入出力）デバイスを管理する
機能です

どのようなことを管理するのか
そこではどんな技法が使われるのか
などを学びます

入出力デバイスとは

- 「デバイス」は「装置」と言い換えてもいいでしょう

入出力デバイスとは

- 「デバイス」は「装置」と言い換えてもいいでしょう
  - 要するに、キーボードやプリンタやディスクやもっと変わった装置も含めて、です

入出力デバイスとは

- 「デバイス」は「装置」と言い換えてもいいでしょう
  - 要するに、キーボードやプリンタやディスクやもっと変わった装置も含めて、です
  - 基本は、全ての入出力装置をOSが管理します
    管理したほうが便利なことがあるからです

OSによる入出力デバイス管理

- どういう管理をするのか？ ⇒ 大きく２点
  - （資源管理）
  - （仮想化）

OSによる入出力デバイス管理

- どういう管理をするのか？ ⇒ 大きく２点
  - （資源管理）

OSによる入出力デバイス管理

- どういう管理をするのか？ ⇒ 大きく２点
  - （資源管理）

１つしかない装置を、複数のユーザ・仕事で共有することになるので、

OSによる入出力デバイス管理

- どういう管理をするのか？ ⇒ 大きく２点
  - （資源管理）
    １つしかない装置を、複数のユーザ・仕事で
    共有することになるので、
    共有資源として「適切に」管理する

OSによる入出力デバイス管理

- どういう管理をするのか？ ⇒ 大きく２点
  - （資源管理）
    １つしかない装置を、複数のユーザ・仕事で
    共有することになるので、
    共有資源として「適切に」管理する
      １）排他的な利用
      ２）効率的な利用・優先利用etc
    詳細は後から説明

OSによる入出力デバイス管理

- どういう管理をするのか？ ⇒ 大きく２点
  - （資源管理）
    １つしかない装置を、複数のユーザ・仕事で
    共有することになるので、
    共有資源として「適切に」管理する
  - （仮想化）

OSによる入出力デバイス管理

- どういう管理をするのか？ ⇒ 大きく２点
  - （資源管理）
    １つしかない装置を、複数のユーザ・仕事で
    共有することになるので、
    共有資源として「適切に」管理する
  - （仮想化）
    種類の異なる装置は、制御方法も違い面倒！
    ユーザ/プログラムから同じように操作したい

OSによる入出力デバイス管理

- どういう管理をするのか？　⇒　大きく２点
  - （資源管理）
    １つしかない装置を、複数のユーザ・仕事で
    共有することになるので、
    共有資源として「適切に」管理する
  - （仮想化）
    種類の異なる装置は、制御方法も違い、面倒！
    ユーザ/プログラムから同じように操作したい
    ⇒　同じ (仮想) インタフェースを提供する

入出力デバイスの共有資源管理

- 資源管理（共有リソースとしての管理）は
  – 管理の２つのポイントがある

入出力デバイスの共有資源管理

- 資源管理（共有リソースとしての管理）は
  - 管理の２つのポイントがある
  - 排他管理
    同時に１人しか使えないように
  - 効率向上（性能向上）
    （１台しかない装置を）効率よく使いたい

入出力デバイスの共有資源管理

- 排他管理　⇒　なぜ必要か？
  - プリンタやキーボードの場合、どうか？

入出力デバイスの共有資源管理

- 排他管理　⇒　なぜ必要か？
  - プリンタやキーボードの場合
    １つしかない装置に、違うプログラムが出力
    ⇒　出力が混ざってしまう？

入出力デバイスの共有資源管理

- 排他管理　⇒　なぜ必要か？
  - プリンタやキーボードの場合
    １つしかない装置に、違うプログラムが出力
    ⇒　出力が混ざってしまう？
    一連の出力が終わるまで占有する必要がある
    当然、他方は待たされる

入出力デバイスの共有資源管理

- 排他管理　⇒　なぜ必要か？
  - プリンタやキーボードの場合
    １つしかない装置に、違うプログラムが出力
    ⇒　出力が混ざってしまう？
    一連の出力が終わるまで占有する必要がある
    当然、他方は待たされる

    実現は、並行プロセスでの排他制御と同様

入出力デバイスの共有資源管理

- 排他管理　⇒　なぜ必要か？
  - プリンタやキーボードの場合
    一連の出力が終わるまで占有する必要がある

  - ハードディスクの場合、どうか？

入出力デバイスの共有資源管理

- 排他管理　⇒　なぜ必要か？
  - プリンタやキーボードの場合
    一連の出力が終わるまで占有する必要がある
  - ハードディスクの場合、どうか？
    １ブロックの読書きの単位で占有すればよい

入出力デバイスの共有資源管理

- 排他管理　⇒　なぜ必要か？
  - プリンタやキーボードの場合
    一連の出力が終わるまで占有する必要がある
  - ハードディスクの場合、どうか？
    １ブロックの読書きの単位で占有すればよい
    むしろスペースが重ならないような管理必要
    ⇒　スペース管理は「ファイルシステム」の章

入出力デバイスの共有資源管理

- 効率向上　⇒　何が問題か？

入出力デバイスの共有資源管理

- 効率向上　⇒　何が問題か？
  - 銀行のATMの行列を思い出してみると
    利用の順番を変えると（スケジューリング）
    性能（例えば平均待ち時間）が変わる
    例えば、処理時間の短いものを優先とか

入出力デバイスの共有資源管理

- 効率向上　⇒　何が問題か？
  - 銀行のATMの行列を思い出してみると
    利用の順番を変えると（スケジューリング）
    性能（例えば平均待ち時間）が変わる
  - ハードディスクの場合
    ⇒　次ページ

（脱線）ハードディスクの性能モデル

- 性能向上技術の説明の前に、
  ハードウェアの授業で見たはずの
  ハードディスクの性能モデルを復習

（脱線）ハードディスクの性能モデル

- ハードディスクのアクセス時間の
  モデルを覚えていますか？
- アクセス時間は、３つの要素の和

（脱線）ハードディスクの性能モデル

- ハードディスクのアクセス時間の
  モデルを覚えていますか？
- アクセス時間は、３つの要素の和

1. アームが動く時間 (別名シーク時間)

（脱線）ハードディスクの性能モデル

- ハードディスクのアクセス時間の
  モデルを覚えていますか？
- アクセス時間は、３つの要素の和

1. アームが動く時間 (別名シーク時間)
2. ディスクの回転待ち時間
   - 欲しいブロックがヘッドの下に来るまでの時間

（脱線）ハードディスクの性能モデル

- ハードディスクのアクセス時間の
  モデルを覚えていますか？
- アクセス時間は、３つの要素の和

1. アームが動く時間 (別名シーク時間)
2. ディスクの回転待ち時間
   - 欲しいブロックがヘッドの下に来るまでの時間
   - 平均すると1/2回転する時間

（脱線）ハードディスクの性能モデル

- ハードディスクのアクセス時間の
  モデルを覚えていますか？
- アクセス時間は、３つの要素の和

1. アームが動く時間 (別名シーク時間)
2. ディスクの回転待ち時間
3. データ (ブロック内容) の読出し・転送時間

（脱線）ハードディスクの性能モデル

- ハードディスクのアクセス時間の
  モデルを覚えていますか？
- アクセス時間は、３つの要素の和

1. アームが動く時間 (別名シーク時間)
2. ディスクの回転待ち時間
3. データ (ブロック内容) の読出し・転送時間

この２者が大半を占める

（脱線）ハードディスクの性能モデル

- ハードディスクのアクセス時間の
  モデルを覚えていますか？
- アクセス時間は、３つの要素の和

1. アームが動く時間 (別名シーク時間)
2. ディスクの回転待ち時間

（この２者が大半を占める）

3. データ (ブロック内容) の読出し・転送時間

この中で、アーム移動時間を減らす工夫を紹介

ハードディスクのスケジューリング

- ブロック読書きの順番を変える
  ⇒　所要時間が変わる

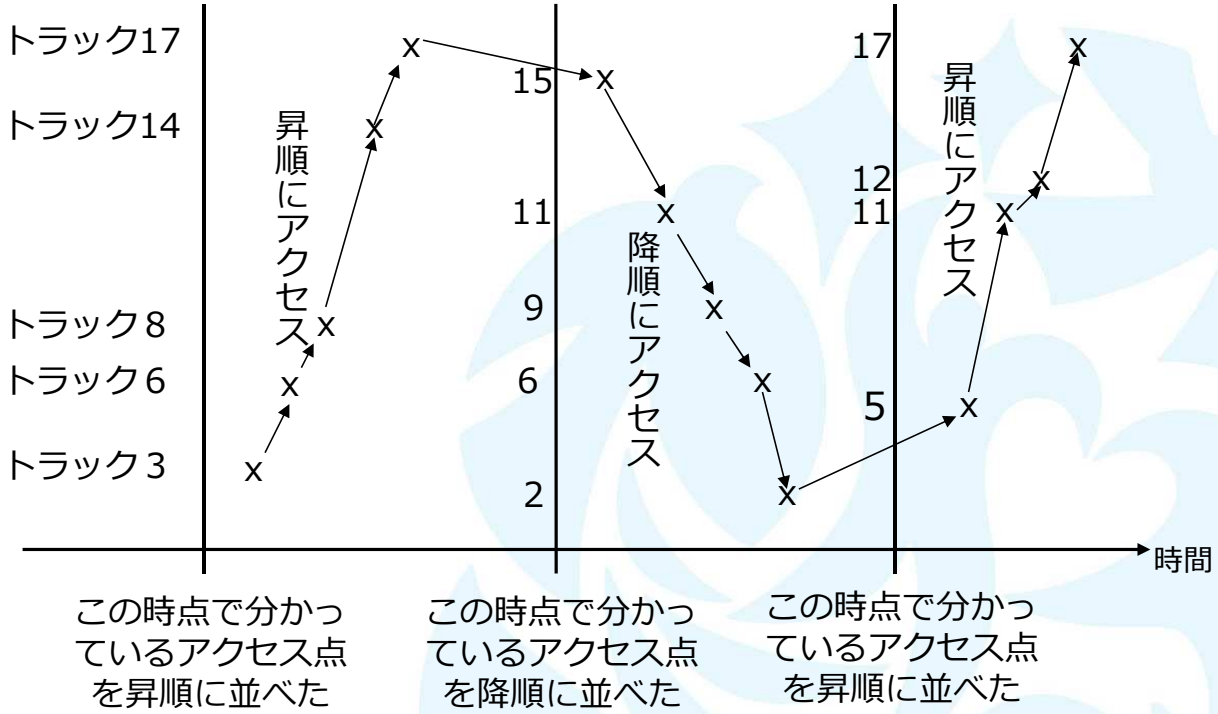
ハードディスクのスケジューリング

- ブロック読書きの順番を変える
  ⇒　所要時間が変わる
- ポイントは
  アーム移動時間が移動距離に依存する
  近いと早い ⇒ 近い移動を優先する

ハードディスクのスケジューリング

- ブロック読書きの順番を変える
  ⇒　所要時間が変わる
- ポイントは
  アーム移動時間が移動距離に依存する
  近いと早い ⇒ 近い移動を優先する
- エレベータ式移動　（教科書p40 SCAN）
  行き先を昇順に順番に登り続ける、頂点で向きを変え降順に降り続ける

ハードディスクのスケジューリング

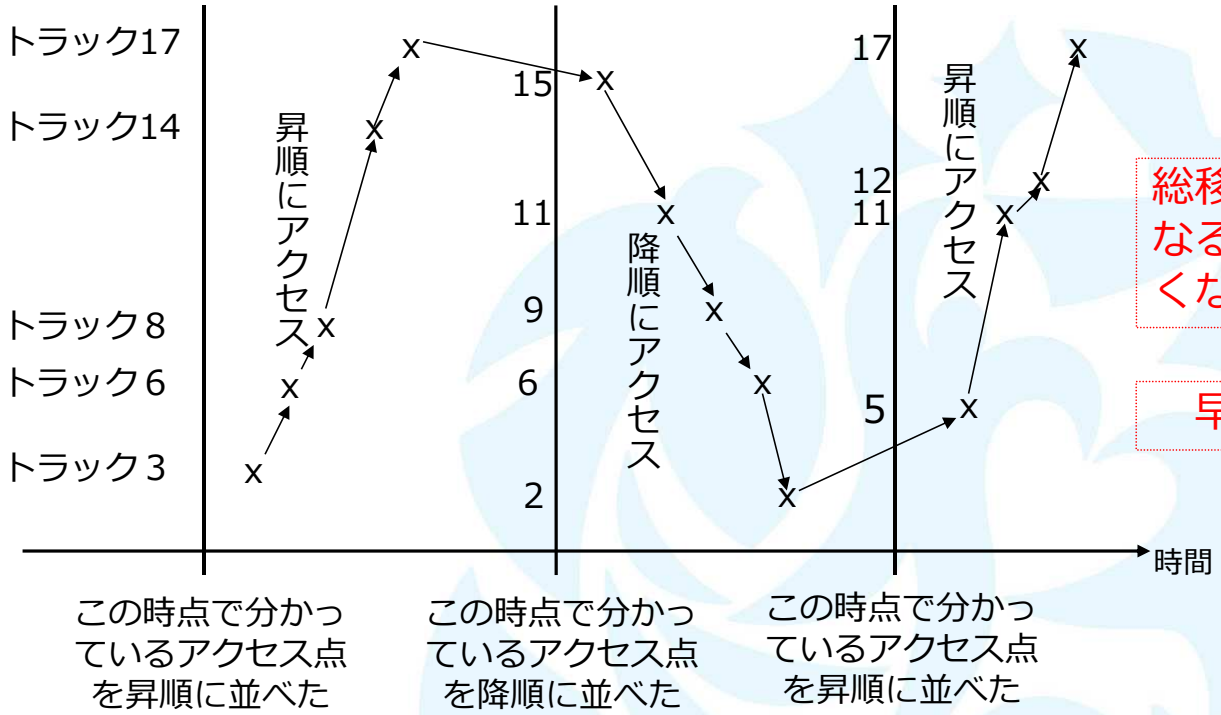


この時点で分かっているアクセス点を昇順に並べた

この時点で分かっているアクセス点を降順に並べた

この時点で分かっているアクセス点を昇順に並べた

ハードディスクのスケジューリング



この時点で分かっているアクセス点を昇順に並べた

この時点で分かっているアクセス点を降順に並べた

この時点で分かっているアクセス点を昇順に並べた

総移動距離がなるべく小さくなるように

↓

早くなる

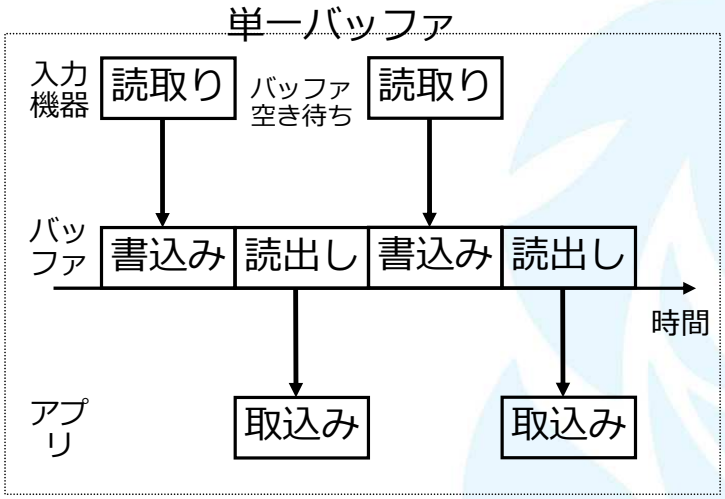
バッファリング（ダブルバッファ）

バッファリング（ダブルバッファ）

- バッファを複数置くことによって
  バッファ空き待ち時間を減らせる

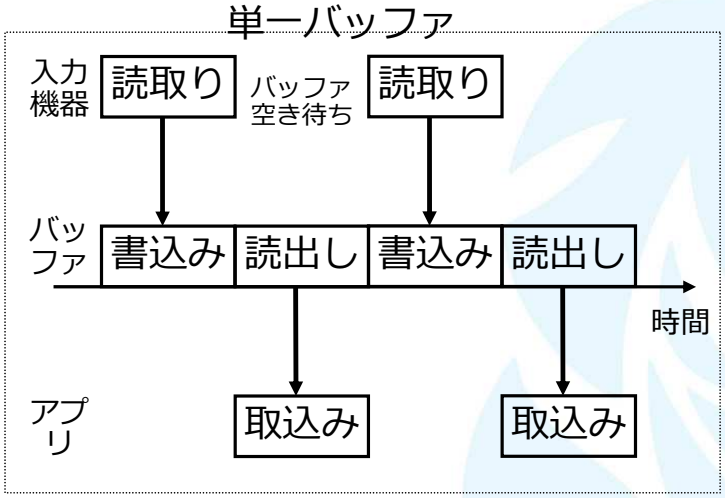
バッファリング（ダブルバッファ）

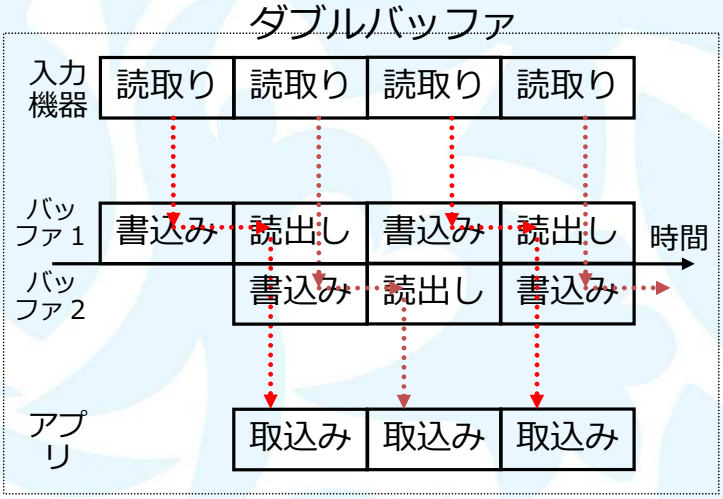
- バッファを複数置くことによって
  バッファ空き待ち時間を減らせる



バッファリング（ダブルバッファ）

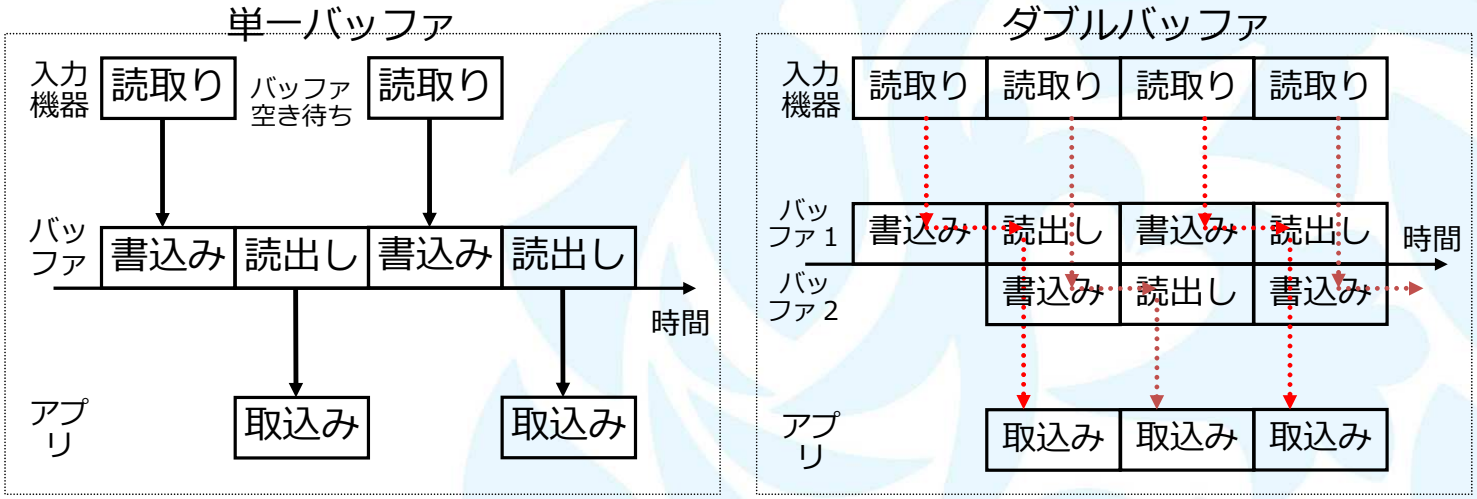
- バッファを複数置くことによって
  バッファ空き待ち時間を減らせる





バッファリング（ダブルバッファ）

- バッファを複数置くことによって
  バッファ空き待ち時間を減らせる



思ったようにうまく行くものか、成り立つ条件をよく考えてみてください

バッファリング（ダブルバッファ）

- バッファを複数置くことによって
  バッファ空き待ち時間を減らせる
- バッファ数は２より大きくできる
  リングバッファ（既出）（教科書p64）

ブロッキング

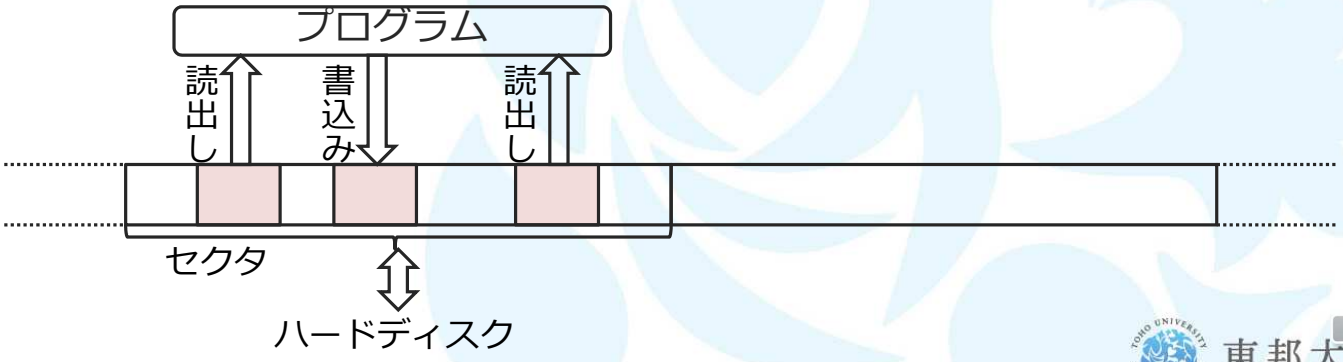
- ブロック化

ブロッキング

- ブロック化　～　複数の入出力操作を
  まとめて１つのブロックにする

ブロッキング

- ブロック化　～　複数の入出力操作を
  まとめて１つのブロックにする
  - ディスクの場合、ハードウェアブロック（セクタ）で読み書きするが、プログラムからの操作は小さい大きさで読み書きする



ブロッキング

- ブロック化　～　複数の入出力操作を
  まとめて１つのブロックにする
  - ディスクの場合、ハードウェアブロック（セクタ）で読み書きするが、プログラムからの操作は小さい大きさで読み書きする
    ⇒メモリ上のバッファへセクタ単位で読出し、
    その上で読み書きして、セクタ単位で書き戻す
    ⇒ハードディスクへの入出力操作が少なくなる効果

ブロッキング

- ブロック化　～　複数の入出力操作を
  まとめて１つのブロックにする
  - ディスクの場合、ハードウェアブロック（セクタ）で
    読み書きするが、プログラムからの操作は小さい大きさ
    で読み書きする
    ⇒メモリ上のバッファへセクタ単位で読出し、
    その上で読み書きして、セクタ単位で書き戻す
  - 磁気テープの場合、起動停止に時間がかかるため
    まとめて読み書きして、起動停止の回数を減らす

(ディスクの) プリフェッチ

- 要るか要らないか分からないが、
  (次のブロックも一緒に) 読んでおく
  - デバイスを１回だけ起動すればよいので、
    次ブロック読出し起動の時間が要らない
    アーム移動（シーク）が要らない
  - （要らなかった時には）無駄になる
  - 例えばファイルを順序に読む場合には有効

ここまでのまとめ

- 入出力デバイス管理には、
- 共有のための資源管理の機能
  - 排他管理
    - 必要な理由？
  - 効率向上
    - 効率向上のテクニック？
- 仮想化の機能

ここまでのまとめ

- 入出力デバイス管理には、
- 共有のための資源管理の機能
  - 排他管理
    - 入出力機器を占有して使う必要がある
  - 効率向上
    - ディスクスケジューリング
      バッファリング・ブロッキング・プリフェッチ
- 仮想化の機能

デバイス管理の考え方と資源管理の内容が掴めましたか？

次へ