授業では、主としてOSの三つの大きな機能を説明して行きますが、今回はプロセスとその管理についてです。

教科書では第7章の「プロセスとその管理」になります。

オペレーティングシステム
（2024年 第４回）

プロセス管理について（1）

前回の課題について

ある割込み処理中にほかの割込みを許す場合、何を注意すべきか

- 割込み事象には優先度が設定されており、優先度の高い割込み処理実行中には、低いものは、そのときは通知されない
  - 優先度の高い割込みが発生したときに、それに対する処理を行うため割込みを許可しておく
- しかし、割込み処理の初期段階に割込みが発生すると処理に矛盾が生じてしまうため、割込み禁止とする期間を設ける
  - 割込み禁止はできるだけ早く解除する
- 割込み処理実行中に、さらに割込みを受付け可能とするには、その割込み処理を呼出す際に現在の実行状態やデータをスタックに保存し、呼出し元へ処理が戻ったときにスタックからデータを復帰するような処理が必要になる

プロセスとは

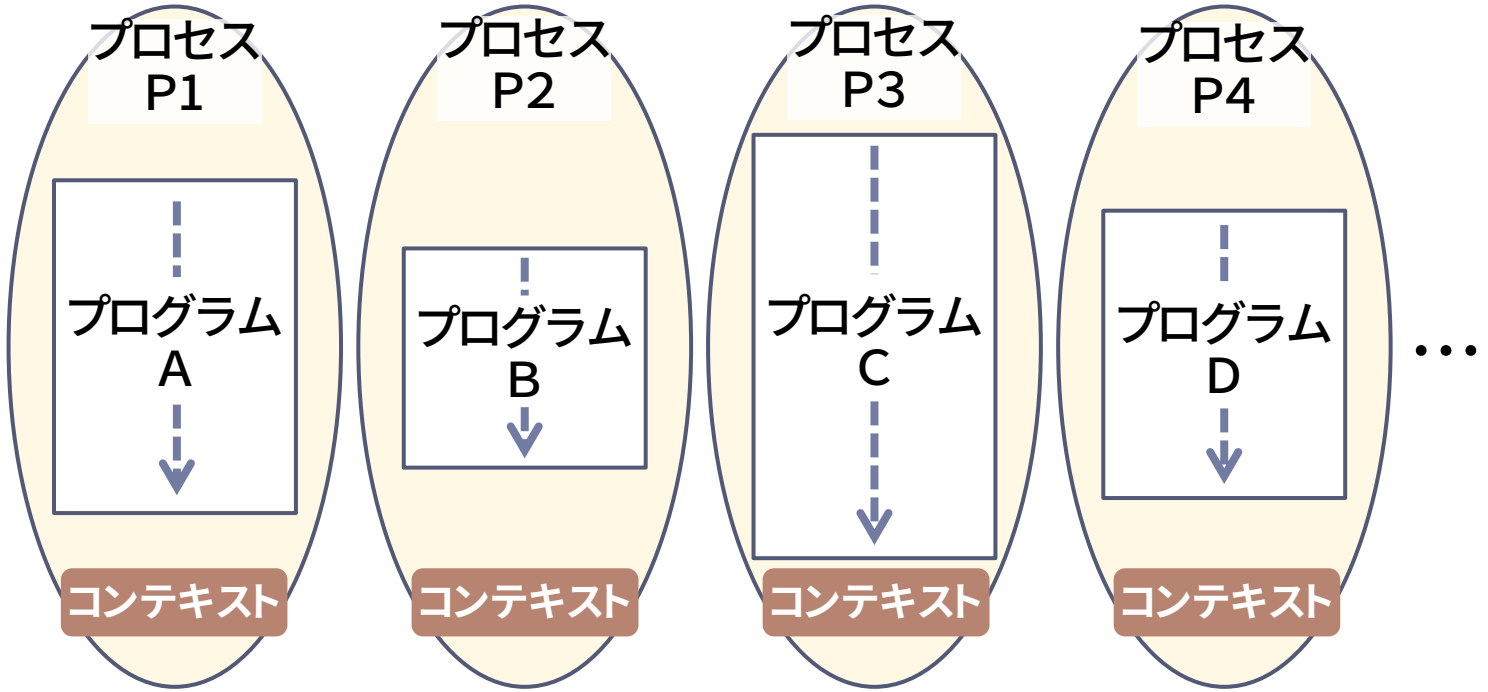
- プロセス（タスク）：プログラム実行中の実体（実行主体）
  プログラムの実行コード、全ての変数、状態 … メモリ内に保持
  - プログラムに対応する命令コード
  - 変数など処理されるデータ
  - プロセスに固有のデータ
    - 実行中のサブルーチン（関数など）の管理領域（コールスタック）
    - ヒープ領域
  - CPUの状態（コンテキスト）
    - PSW、レジスタの内容など
  - プロセスに割当てられたファイルなどの資源の記述子
  - プロセスの所有者やプロセスに関わる権限などの情報
- すべてのアプリケーションプログラムはプロセスとして実行される

プロセスとは、プログラムの実体で、実行される命令の系列と操作の対象となる変数などのデータ、そして実行中の状態からなるもので、プログラムが動いているときの状態も合わせて保持しているものと考えられます。

具体的には、以下の要素がメモリに格納されます。
変数など実行によって値が変わって行くもの、関数がどこから呼ばれたかなど実行時の経路情報なども状態を作るものです。また、CPUの実行状態やレジスタなどのリソースの情報も含まれ、これは特にコンテキスト（プロセッサコンテキスト）と呼ばれます。

プロセスの概念に対する詳しい説明を別資料としましたので読んでおいてください。和訳洋書の参考書「オペレーティングシステムの概念」の第3章からの引用です。

- それぞれ独立した処理の流れ
  - 逐次に実行して行く
    プロセスには1つのプログラムカウンタ(PC)が対応する



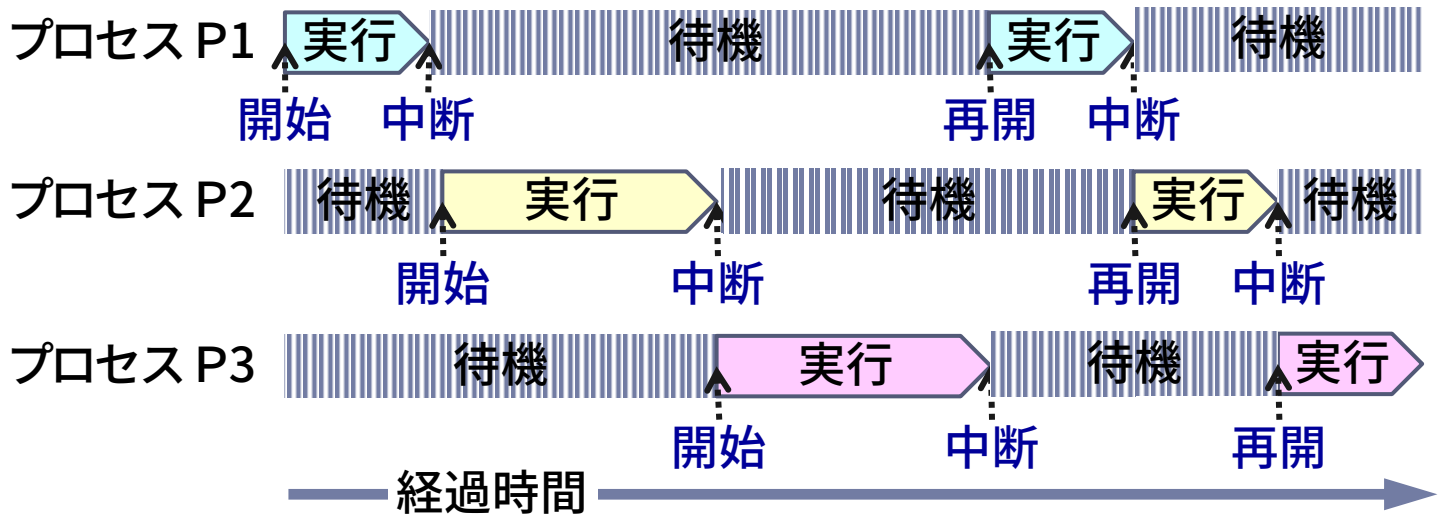
  - ある時点ではP1としてAを実行、別の時点ではP2としてBを実行 …

プロセスとはすなわち、プログラムが動作するときの必要なリソースをまとめたものです。
最初の講義でお話ししましたが、命令とデータがメモリの中にあります。プログラムを実行するには、そこで説明した管理用の固有データも必要で、それらがまとまったものとしてメモリに配置され、複数のプロセスではそれぞれが独立しているものとみることができます。

通常、複数のユーザのいろいろなプログラムが同時に動く状態にあるものと考えることができ、この図のようにプログラムに対するプロセスが独立して複数あります。
CPUは、ある時点ではプロセスP1としてプログラムAを実行し、コンテキストを始めする状態を変化させ、保持しています。別の時点では、P2としてBを実行する・・・となります。

マルチプロセス

- 複数のプロセスを生成でき、CPUが1つでも見かけ上、それらが同時に実行される
- プロセッサの時分割利用
  - 実在するCPUが1つであっても複数のように見せる … CPUの多重化
  - イベントにより、実行するプロセスを切り替える
    - ある時点でCPU画はプロセスP1を実行
      イベントの発生で別のプロセスP2を実行、このときP1の実行は中断される
    - 別の時点で実行中のプロセスが中断し、P1の実行が再開

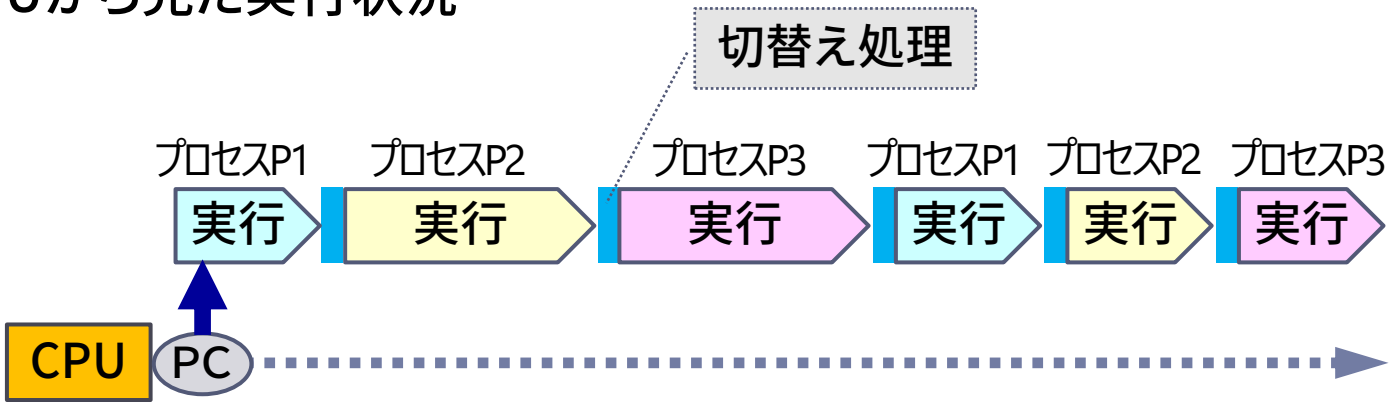


このようなことがマルチプロセスであり、CPUは1つだけでも、ある時点ではあるプロセスを実行していて、あるきっかけで別のプロセスを実行するということになります。
その際、それまで実行していたプロセスは中断されます。
中断されていたプロセスは、また別のきっかけで再開されることになります。

このきっかけ(イベント)が前回の割込みや例外を使用して実現されます。

プロセスの切り替え

- 1つのCPUしかないコンピュータでは、ある瞬間には1つの処理しか実行できないが、CPUの処理時間を数十ミリ秒といった短い時間毎に区切り、複数のプロセスで1つのCPUを交互に使用（時分割）することによって、ユーザから見ると複数のプロセス（アプリケーション）が同時に実行されているように見える
- プロセス切替えのオーバーヘッドなどのコストがかかるが、入出力待ちなどであるプロセスの実行が止まっても他のプロセスが実行されるため、全体としてスループットの上昇が期待できる
- CPUから見た実行状況



CPUが1つの場合、ある瞬間には1つの処理しかできませんが、数ミリ秒とか数十ミリ秒の時間で、現在実行中のプロセスを中断して別のプロセスを動作させることを繰り返すと、複数のプロセスが同時に実行されているように見えます。
資源管理の着想で挙げた「時分割」を使って、CPUの仮想化、すなわちたくさんあって、複数のプロセスを同時に実行することが実現されます。
プロセスを切り替えるにはオーバヘッドが発生しますが、入出力を待つ間に他のプロセスを実行できて、全体としての処理時間が短くできることや、複数プロセスが実行できるといったことによるメリットが大きいといえます。

CPUから見た場合は、切替え処理も含めた異なるプロセスの部分・部分を逐次実行しているイメージになります。（すべてが連続したアドレスでならべられているわけではありません）

プロセッサコンテキストとプロセス切り替え

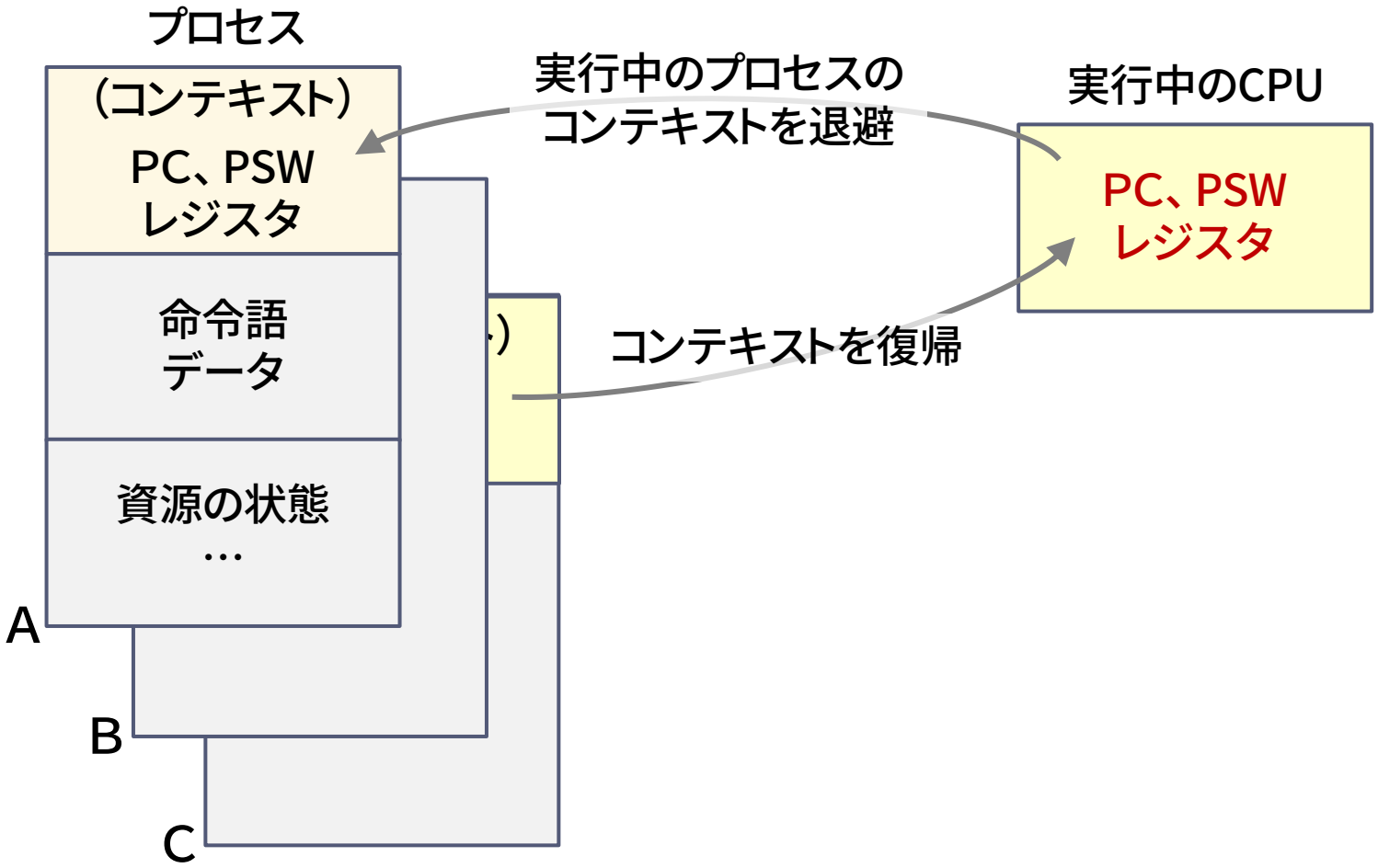
- プロセッサコンテキスト
  プログラム実行時のプロセッサの状態、特にCPU内のレジスタの値
  - プロセスを切り替える際、プロセスの状態を保持し、他のプロセスを実行後に正しく再開できるようにする
  - 実行されていないプロセスのプロセッサコンテキストは主メモリ上にある
- プロセスの切替え
  プロセッサコンテキストをCPUとメモリの間で転送しあうことで、プロセスを切り替えて実行する

中断と再開、すなわちプロセスが切り替わることがどのようにしてなされるかについて説明します。
ここで重要な機構がコンテキスト(プロセッサコンテキスト)です。
p.3の「プロセスとは」のところで、プロセスを構成する要素を説明しましたが、その中にCPUの状態(コンテキスト)―レジスタの内容など、というものがありました。

プロセスを切り替えるためには、中断したプロセスを中断点(の次)から再開できることが必要です。命令語とデータはメモリにあり、管理データもメモリにあります。
すなわち、これらは、他のプログラムが動作した後でも、(異なるメモリ上にあれば)残っていて、再開のためにそのまま利用できます。
そこで、CPU内のレジスタの状態、PCや汎用レジスタ、PSWなど、が保存されていればよいのではと考えられます。これらはCPUで1セットしかないので、別のプログラムが動作するとその実行で書き換わってしまいます。

中断点で、その時のコンテキスト(上記のレジスタなど)を自分のプロセスのメモリ内に書いて保存(退避などとも言います)しておけば、再開時にそこから回復すれば、中断点の次から実行できるとなります。

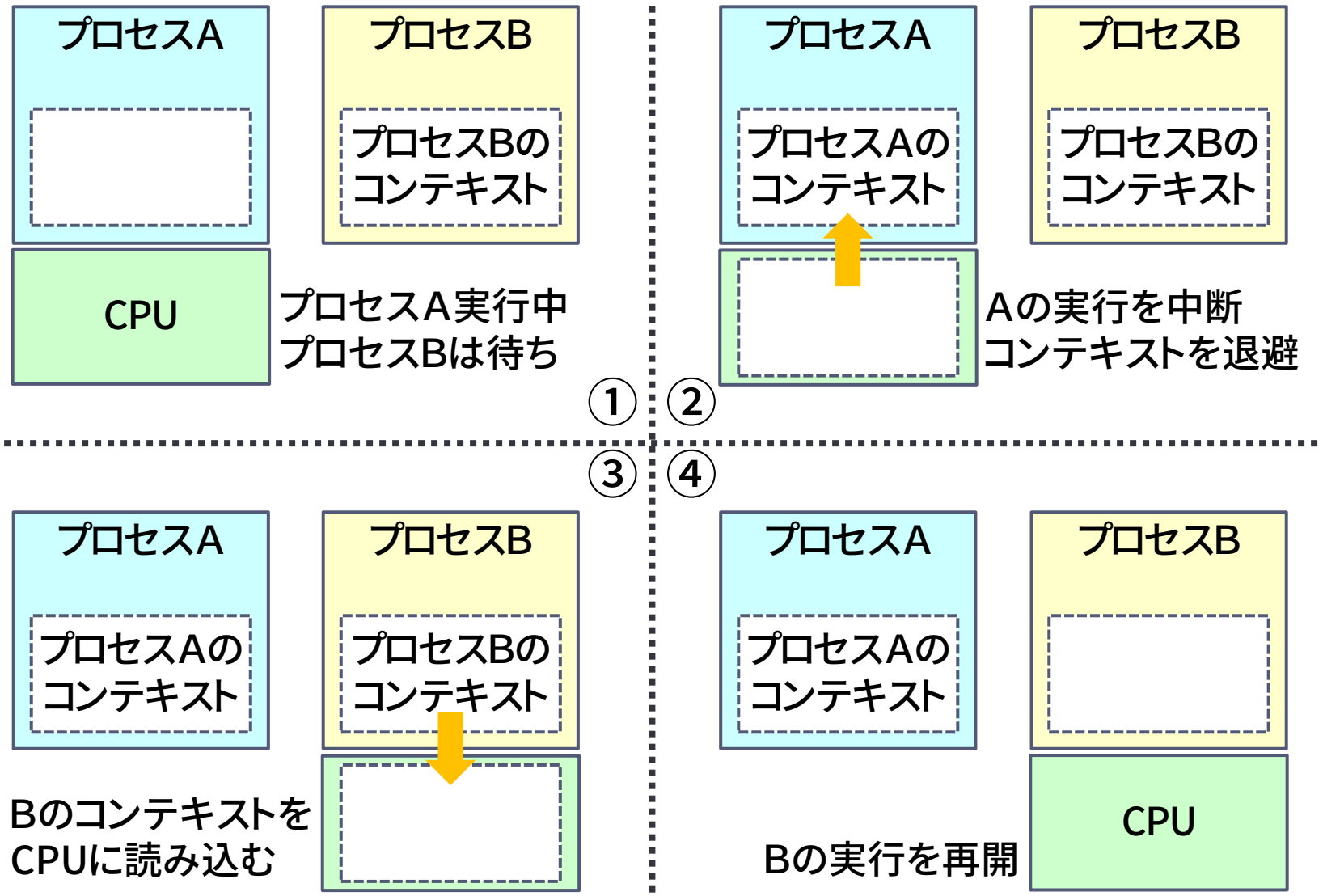
プロセス切り替えのイメージ



プログラムは、実行中にレジスタの値を書き換え、メモリの値を書き換え、順次進行していきます。ここで、メモリはプロセスごとに固有の領域があります。
しかし、レジスタはすべてのプロセスで共有されるものです。

そこで、レジスタなどの値を個々のプロセスのメモリの中に持っていればよいと考えられるわけです。プロセスの中にPCや汎用レジスタなどといったCPUのリソースの値を保持する領域を用意することになります。

プロセスの切り替え（中断と再開）の動作

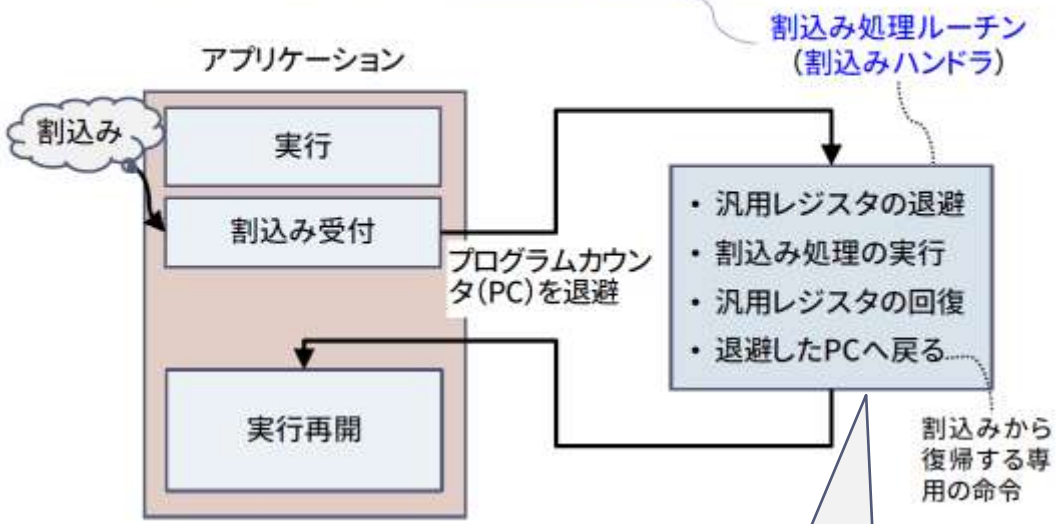


退避されていたコンテキストを回復することで中断時点の実行状態を再現でき、回復されて次に実行すべき命令を指しているPCから命令を解釈実行することで、再開されます。

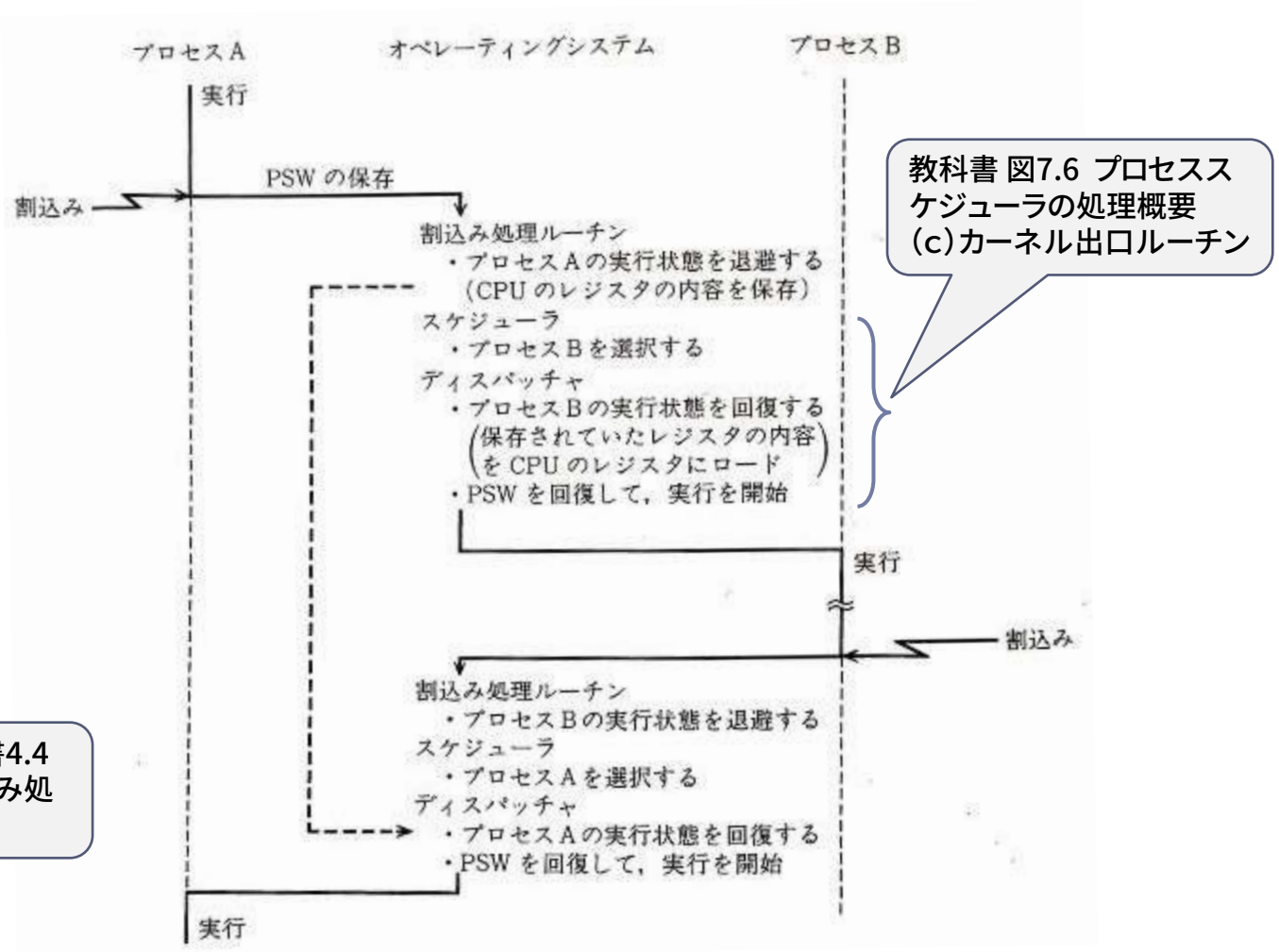
図で、③がコンテキストの回復で、その前に②で中断するプロセスのコンテキストを退避しています。

プロセス切り替え処理

- 割込みを受付けるとCPUは特定の番地に制御を移行する
  その番地から始まるように割込み用の処理を置いておく



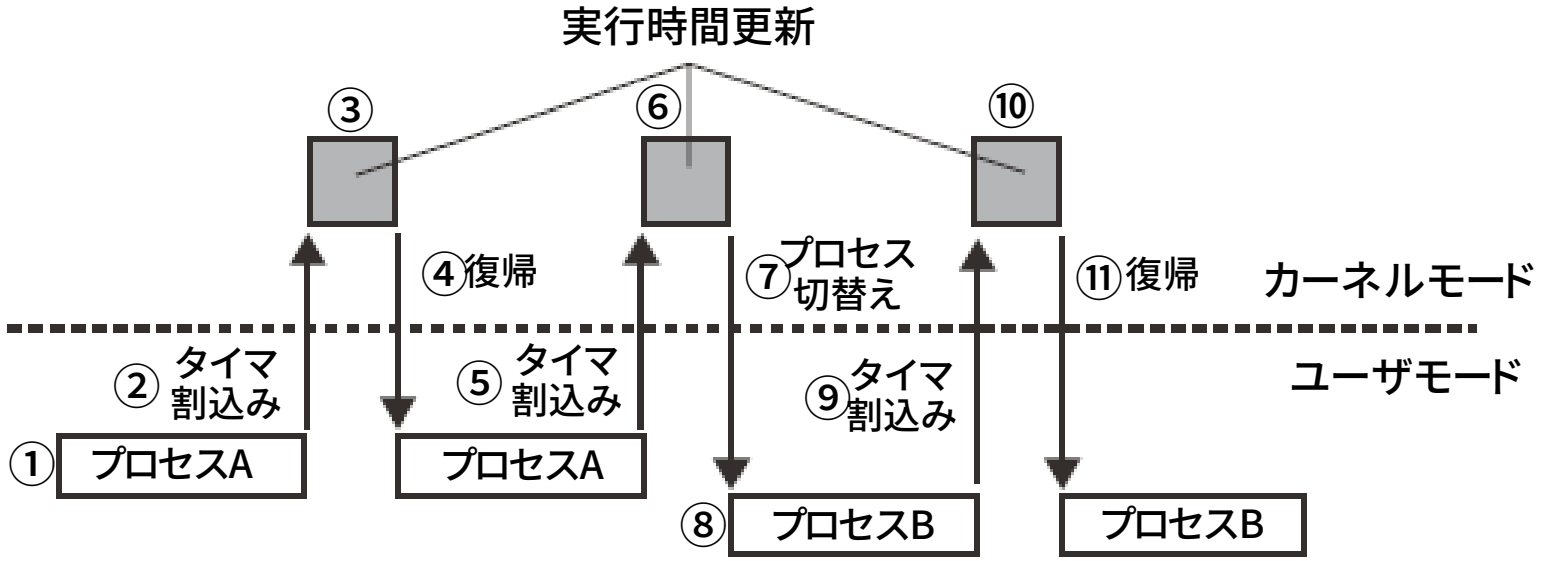
詳細な処理内容は、教科書4.4 カーネルへの入り口：割込み処理



教科書 図7.6 プロセススケジューラの処理概要
(c)カーネル出口ルーチン

参考書「オペレーティングシステム」より図6.1 プロセスの切替え

例：タイマ割込みとプロセスの切り替え



① ユーザモードでプロセスAが実行

② タイマ割込みが発生、カーネルに制御が移る

③ タイマ割込みの処理でプロセス利用時間を更新

④ 割り当てられたCPU時間を使い切っていないので、Aに復帰し実行再開

⑤ タイマ割込みが発生、カーネルに制御が移る

⑥ タイマ割込みの処理でプロセス利用時間を更新

⑦ 割り当てられたCPU時間を超えていたのでプロセスを切り替える（Bが選択される）

⑧ Bに復帰し、Bの実行を再開

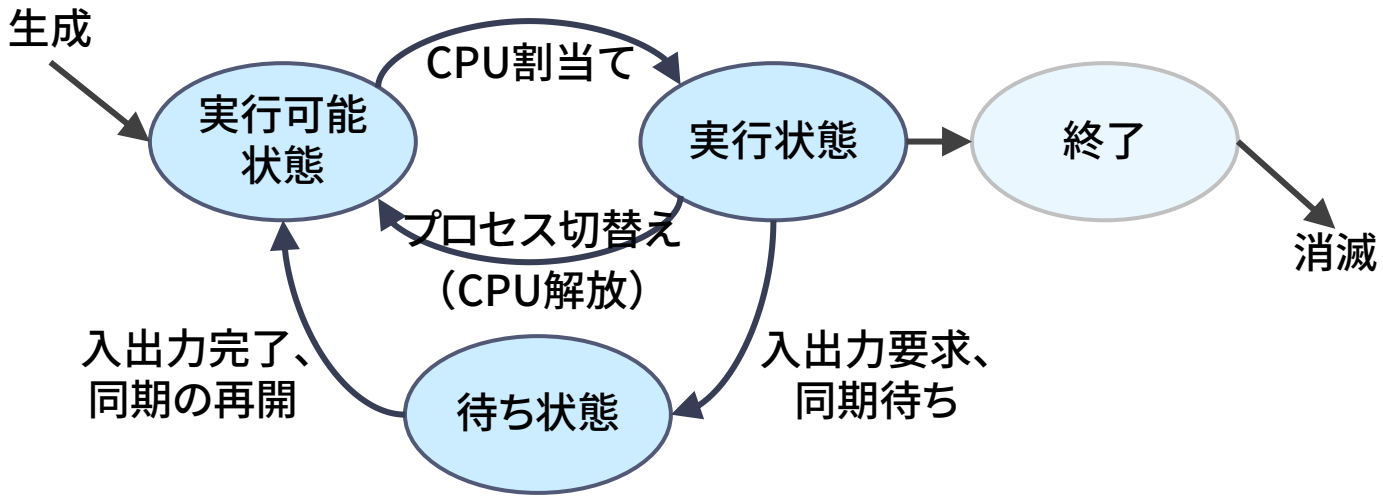
⑨ タイマ割込みが発生、カーネルに制御が移る

⑩ タイマ割込みの処理でプロセス利用時間を更新

⑪ 割り当てられたCPU時間を使い切っていないので、Aに復帰し実行再開

プロセスの状態と遷移

- プロセスの状態と遷移
  - CPUが割り当てられて実行されている状態
    - 実行状態（running）
  - 実行が一時中断されている状態
    - 実行可能状態（ready）… CPUが割り当てられれば実行できる状態
    - 待ち状態（waiting）… 入出力の完了や他のプロセスとの動機などのイベントを待っている状態
  - 終了した状態
    - 終了（zombie）… プロセスの処理が終了し、資源を解放した状態



プロセスは実行されたり中断されたりします。そこで、1つのプロセスに着目して、それがとる状態というものを考えます。
プロセスがどのような状態であって、それをどのような状態にするかを管理することがプロセス管理の基本です。

CPUが1個だとすると、ある時点で1つのプロセスが実行されていて、それ以外は何らかの理由で実行できず、何らかのきっかけで実行されるようになります。
すなわち、実行中、実行可能、待ち、という3状態が基本的に存在することになります。

実行可能状態と待ち状態の違いは、CPUが空いていても待ち状態のプロセスは実行できないということです。
待ちが解除されると、すぐに実行状態に移るのではなく、実行可能状態に移ることに注意してください。

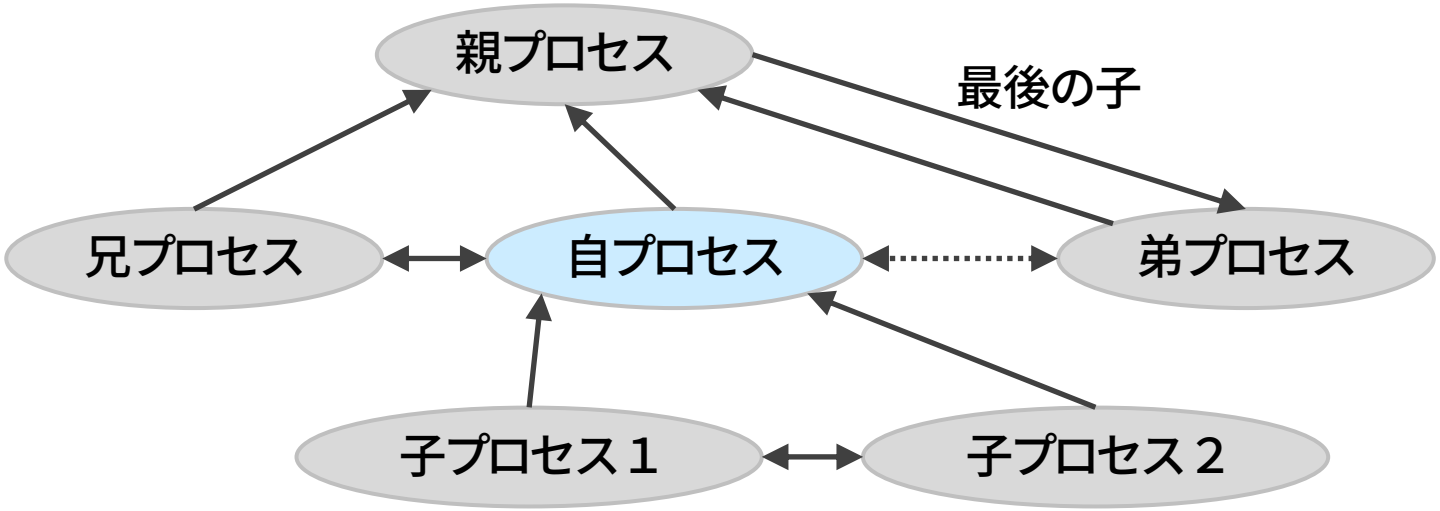
プロセス制御ブロック、プロセス間の関係

- プロセス制御ブロック（Process Control Block：PCB）
  （ プロセス構造体 とも呼ぶ）
  - プロセスの実行に必要な情報を保持するデータ構造
  - プロセッサコンテキスト、プロセスに割り当てられたメモリの情報、プロセスが利用している入出力装置の情報など をもつ
  - 1つのプロセスは1つのプロセス制御ブロックを持ち、メモリに格納される
  - OSはPCBを操作することでプロセスの制御を行う

プロセスを管理するためのデータ構造が必要になります。メモリの中にデータ構造を作って、その中にプロセスを管理するデータを保持するわけです。
先ほどの「プロセスの状態」も管理データの一つです。

プロセス間の関係

- 親子関係 … プロセスにはそれを生成したプロセスが存在する
  プロセスを生成した方が「親プロセス」、
  生成された方が「子プロセス」
- 兄弟関係 … 同じ親プロセスから生成された子プロセス同士



プロセスには親子関係があります。
たとえば、Windowsでフォルダをたどったりファイルを開いたりするエクスプローラーは1つのプロセスです。エクスプローラーで、テキストファイルをダブルクリックして開くと、(たいていは)メモ帳が起動してそのファイルが開きます。これもプロセスです。このとき、エクスプローラーのプロセスが親となり、メモ帳のプロセスは、その子です。この状態で、エクスプローラーからはまた別に、pdfファイルを開くこともできます。Adobe Readerなどが起動してそのpdfファイルが開きます。これもプロセスで、エクスプローラープロセスの子です。
このとき、メモ帳プロセスとAdobe Readerプロセスは兄弟プロセスであり、メモ帳が兄で、Adobe Readerプロセスが弟になります。

あるプロセスを自分(自プロセス)としてみた場合、親、兄弟がそれぞれたどれるようになっています。

プロセス制御ブロックの内容

- 一般的に、直接/間接に以下のような項目を保持する
  （間接とは、その項目へのポインタのこと）
  - プロセスを識別する番号（プロセス識別子：PID）
  - コンテキスト … プロセスのその時点のPSWやレジスタ群、特にプログラムカウンタ（PC）の値
  - オープンしているファイルの情報
  - そのプロセスのアドレス空間やメモリの情報
  - スケジューリングに関する情報
  - シグナルに関する情報
  - プロセスの親子関係情報
  - プロセスのアカウンティング情報
    プロセスが最近動作した時刻、これまでに消費したCPU時間の合計など

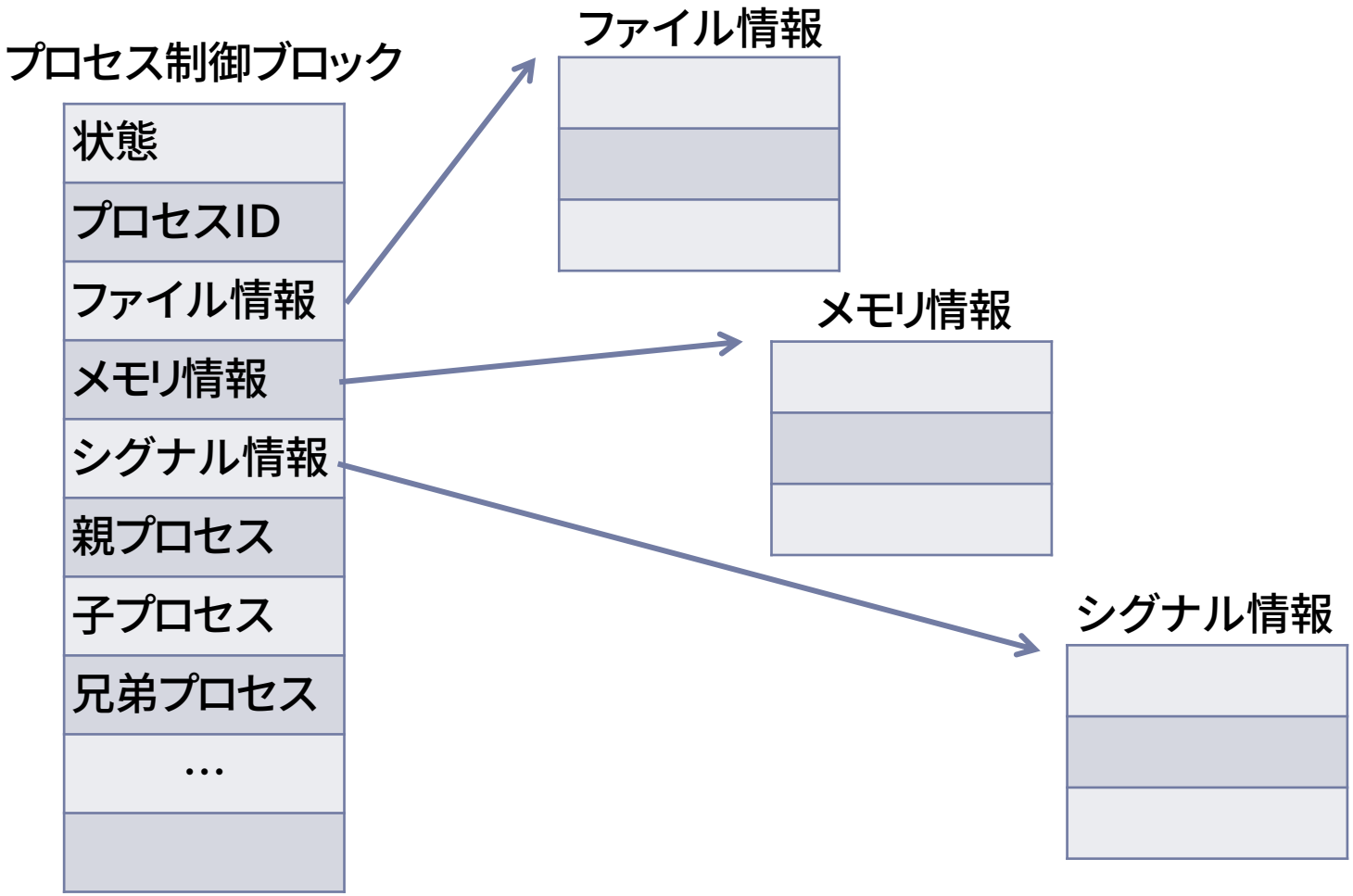
中断時のPCの値から再開する

シグナル … プロセス間で連絡を取り合うための仕組み
割り込み、例外、プロセス終了のイベント、など

このような管理を行うために、プロセス制御ブロックの内容は、ここに示すようになっています。

次ページの図のように構造体があり、そこからさらに別の構造体にリンクが張られているような形式です。

プロセス制御ブロックの内容の例



図で表現すると、このようなイメージです。このような構造をたどって、どこかの要素を見て何かの処理を行うということをOSはやっているわけです。

プロセスのスケジューリング

- スケジューリング
  複数の仕事（プロセスなど）があるときに、どういう順でそれらを進めれば最も都合が良いか（速いか）という最適な順番を求めること
- なぜスケジューリングが必要か
  複数の実行可能なプロセスが同時に1つのCPUを獲得しようとすると、どれか1つのプロセスを選んで次に実行すべきものとして決めなければならない
- スケジューラ
  上記の機能を担当する部分
- スケジューリングアルゴリズム
  選択する方法（決め方）

それでは、そのやっていることの基本です。OSの重要な役目、プロセスにCPUを割りてるスケジューリングです。

複数のプロセスが存在しているので、その中で実行できるものから1つを選んで、それにCPUを割り当てなければなりません。先に挙げたプロセスの状態でいうと、「実行可能」なものから何らかの基準で1つを選択するわけです。

プロセススケジューリング

- プロセススケジューラ（単に スケジューラ とも）
  プロセスの実行を制御するOSのプログラム
  以下を行う
  - ディスパッチ
    「実行可能状態」のプロセスを「実行状態」にし、プロセス制御ブロック内のコンテキストをCPUに格納する（そのプロセスに制御を渡す）
  - プロセススケジューリング
    プロセスの実行を一時中断すると、次に実行するプロセスを選ぶ
    - 余計な処理が不要となるように次のプロセスを選ぶ

  割込み処理と並んでOSの根幹となるプログラム

プロセススケジューラと呼ばれるプログラムがその機能を行います。大きく以下の二つの機能があります。

ディスパッチは、実行可能状態のプロセスを実行状態にすること、すなわち、そのプロセスのプロセス制御ブロックにあるコンテキストをCPUに格納することになります。

現在実行されているプロセスが中断したときに、次にどのプロセスをディスパッチするものとして選ぶかを決めるものがプロセススケジューリングです。

スケジューラの動作

- 実行可能状態のプロセスがキューになっている … 実行可能リスト
- 待ち状態のプロセスもキューになっている … 待ちリスト

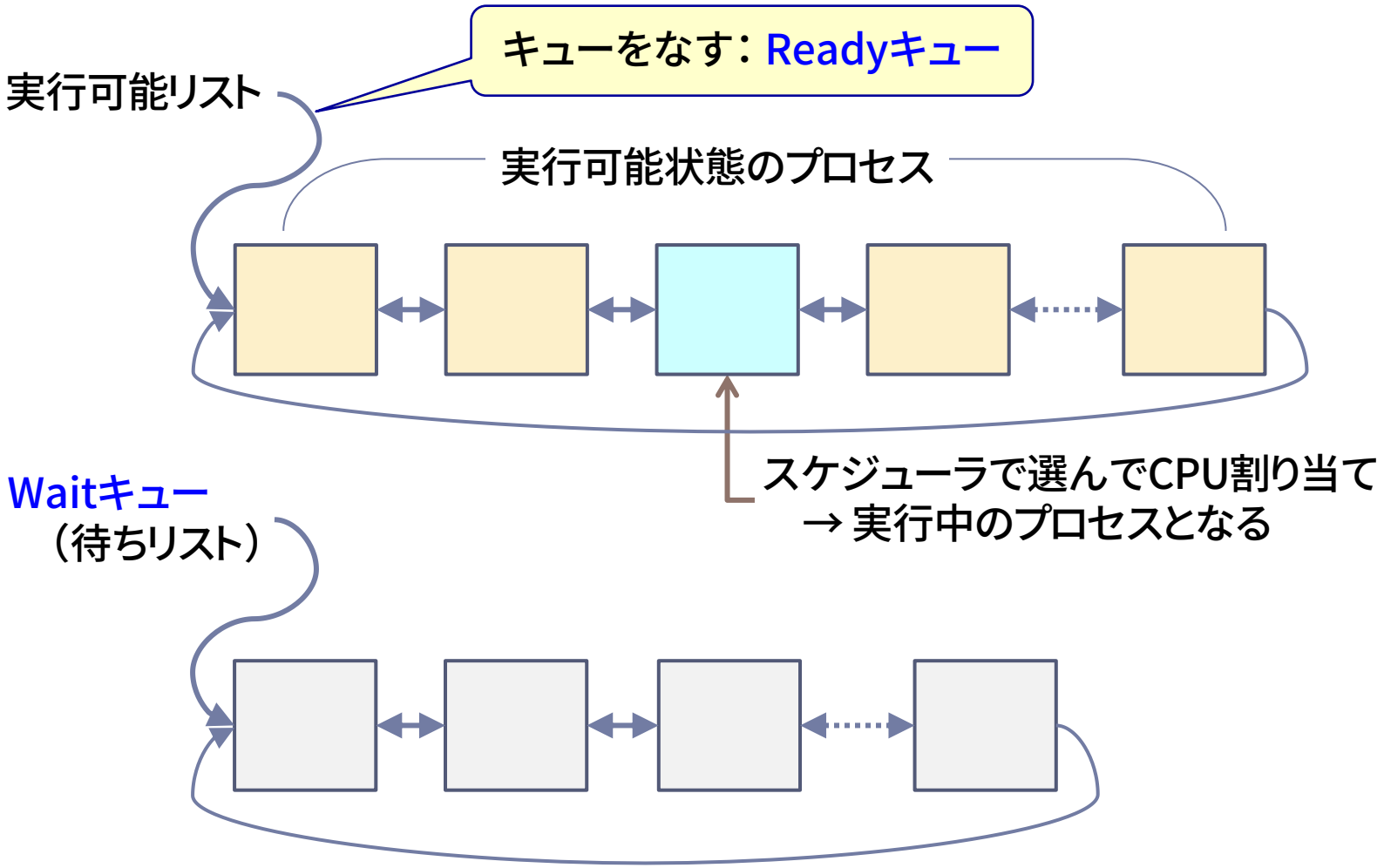
- 実行中のプロセスに待ちが発生すると、プロセススケジューラはそのプロセスを「待ち状態」にし、実行可能リストからそのプロセスを外して待ちリスト（Waitキュー）につなぐ
- その時、最も適切な（優先度の高い）プロセスを「実行状態」にする
- 待ちの原因が解消すると「実行可能状態」になり、実行可能リスト（Readyキュー）につなげられる

スケジューラの動作の概要です。
まず、スケジューラが使うデータ構造ですが、実行可能状態のプロセス、待ち状態のプロセスそれぞれ複数をリストで保持しています。

現在、実行中のプロセスに「待ち」（入出力処理を行ったなど）が発生すると、そのプロセスは待ちリストにつながれます。そして、その時点で最も適切なプロセスを実行状態にします。

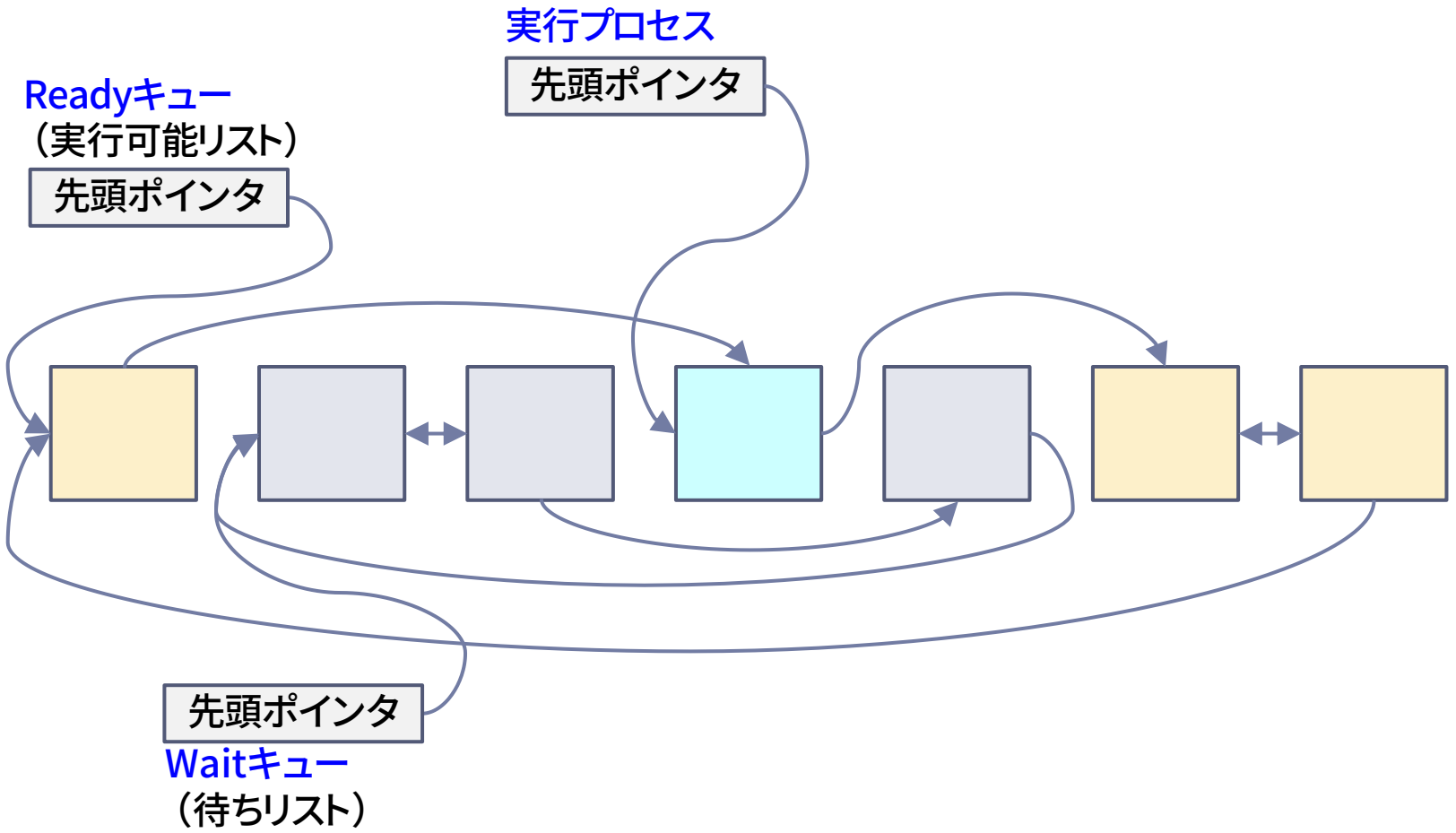
待ちの原因が解消する（入出力完了など）と、そのプロセスは実行可能リストにつなげられます。

プロセススケジューラが使うデータ構造



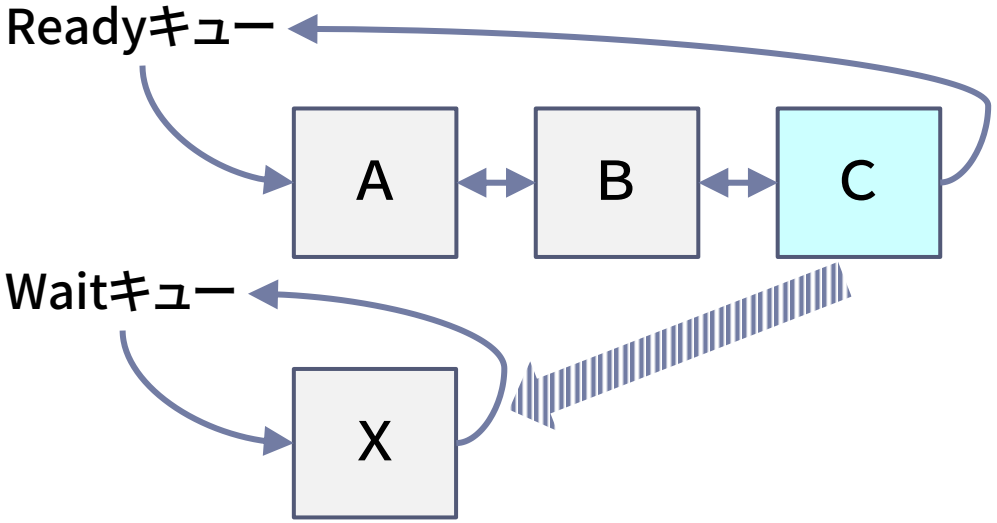
リストのイメージを示します。
実行可能リスト（Readyキュー）や待ちリスト（Waitキュー）は、実行可能状態のプロセスが要素としてポインタでつながったリストとして構成されています。また、前にも後ろにもたどれる双方向のリストで、最後のものから先頭につながる循環リストで実現されることが多いです。
いくつあるかあらかじめわからないもの、要素を挿入したり、削除したりするのに適したデータ構造です。
実行可能リストの中の1つがスケジューラで選ばれて、それが実行状態のプロセスとなります。

実行可能リストの中の1つが現在実行中のプロセスとして管理されます。

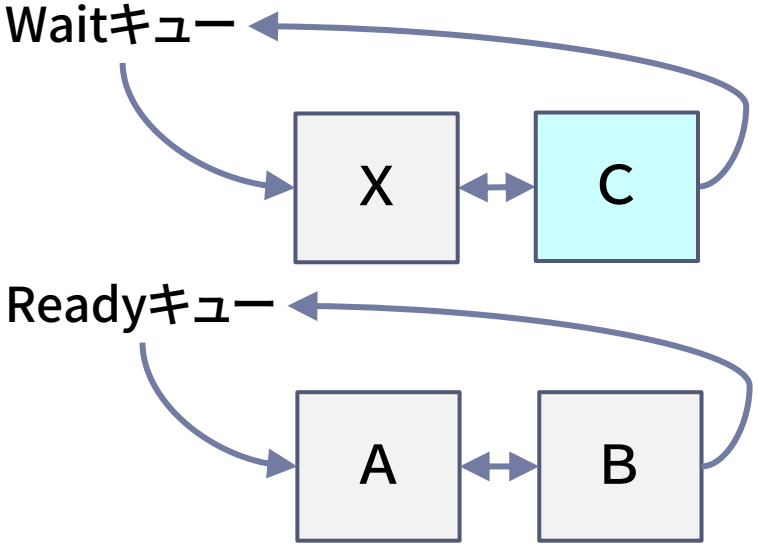


プロセス制御ブロックが並んでいることをイメージします。それぞれ、親や兄弟のリンク（ポインタ）でもつながれていますが、煩雑になるのでここでは示していません。
それぞれにポインタによってつながれて、Readyキューとしてたどれるもの、Waitキューとしてたどれるものとなります。

スケジューラ動作のイメージ



① プロセスC が実行中
② プロセスC がキー入力待ちになる
③ スケジューラがC の状態を「待ち状態」に変更する
④ スケジューラがC をReadyキューから外す



⑤ スケジューラがC をWaitキューにつなぐ

⑥ Readyキューで優先度が最も高いプロセスを「実行状態」にする

このデータを使用して、プロセスが切り替わる時の動作を示すと図のようになります。プロセスCがWaitキューにつながる様子をP.19の記述と照らして確認してください。
⑥でプロセスAかBのどれかが選択されて実行状態になります。

スケジューリングの目標とポリシー

- スケジューリングの目標
  - 処理効率（スループット）の向上
  - 応答時間の短縮
  - 公平な割り当て
  - 応答時間が予測可能

  相反する項目や何を重視するかで対応が異なる
- スケジューリングのポリシー

  最適であることよりも簡便なアルゴリズムを使うことが多い
  - どのようなポリシーでスケジュールするかが重要
    - すべてのプロセスが平等に実行される
    - ある特定のプロセスについて、短時間のうちに実行を終了させる
    - 指定されたプロセス群が指定された時間内に必ず終了することを保証する（リアルタイム性）

    などが考えられる

それでは、どのような順序となるようにスケジューリングするかですが、スケジューリングの目標は以下のようなものがあります。速くするということ、これには、全体として効率が上がるということや、素早く応答できるといったこと、個別のものが早く終了できるといったことなどがあります。
別の観点では、どのプロセスも公平に実行されたいといったことなどがあり、すべての条件が満たされることはありません。したがって、何を重視するかというポリシーに従ってスケジューリングを行います。

OSでは、事前にプログラムのすべての挙動がわかるわけではないこと、スケジューリング問題は組み合わせ問題になることが多く、アルゴリズムによってはスケジューリングのための計算コストが高くなることがあるので、最適であることよりも簡便なアルゴリズムが使われることが多いようです。

スケジューリングアルゴリズム

- プリエンプション（preemption）
  あるプロセスの実行を途中で中断（横取り）し、別のプロセスの実行を開始すること
- 非プリエンプティブスケジューリング（横取りしない）
  - 到着順 … 新たに実行可能になったプロセスから順に処理（CPUを割当てる）
    実行可能になったプロセスをReadyキューの最後につなげる
  - 処理時間順 … 処理時間の短いプロセスから順にCPUを割当てる
    （一般には不可能なので、見積り時間や予測を利用）
- プリエンプティブスケジューリング
  - ラウンドロビン … 短い一定時間（タイムスライス（タイムクゥオンタム））ごとに順繰りにプロセスの実行を切替える
    - 一定時間が経過したプロセスを実行状態から実行可能状態にし、Readyキューの最後につなげる（どのプロセスにも一定のCPU時間を与えられる）
  - 優先度順 … 各プロセスに優先度を与え、この優先度に従って実行する
    Readyキューのプロセスを優先度でソートする

スケジューリングのアルゴリズムで、重要な考えに「プリエンプション」があります。実行を途中で横取りして、別のプロセスを開始することを言います。

プリエンプションがないスケジューリング（非プリエンプティブスケジューリング）には、到着順、処理時間順などの方式が代表的なものとしてあります。
プロエンプティブなスケジューリングではラウンドロビンや優先度順などがあります。
それぞれのスケジューリング方式について、以下のページも確認してください。

種々のスケジューリングアルゴリズム(1)

名　　称	内　　容
FCFS：到着順 First-Come, First-Served First In, First Out	到着したプロセスから順にCPUを割当てる 利点：単純で実現しやすい。資源割付けが公平 欠点：CPUを独占するプロセスが1つでもあれば、その後に待つプロセスは長時間待たされる • 非プリエンプティブ
SPTF：処理時間順 Shortest Processing Time First	処理時間の短いプロセスから順にCPUを割当てる 利点：プロセス実行完了までの時間の平均値が最小になる 欠点：実現が困難（各プロセスの実行時間をあらかじめ知るのは一般には不可能） • 非プリエンプティブ
SRPT：残余処理時間順 Shortest Remaining Processing Time First	実行可能になるたびに、その時点で残っている処理時間の小さいプロセスから順にCPUを割当てる 利点：長くかかるプロセスの完了まで他のプロセスを待たせない 欠点：実現が困難 • プリエンプティブ

FCFSでは、プロセスはいったん実行を開始されると完了するまで実行されます。

SPTFはSPTとも略されます。

SRPTはSPTFと同様の理由で実現は困難です。現在までの実行時間が大きいほど完了までの実行時間も大きいという統計的な性質を利用する方法もあります。

種々のスケジューリングアルゴリズム(2)

名　　称	内　　容
優先度順	利点：優先度の高い（重要な）プロセスが早く実行されるという意味で、効率が良い 欠点：優先度の低いプロセスがいつまでも実行されないこと（飢餓状態：Starvation）が起こり得る • プリエンプティブ
ラウンドロビン	利点：どのプロセスにも一定のCPU時間が与えられ、公平に実行される（タイムスライスは10～100ミリ秒程度） 欠点：実行切替えが頻繁にあり、オーバーヘッドが大きい • プリエンプティブ
締切順スケジューリング (EDF) Earliest Deadline First	プロセスごとに期限（デッドライン）を決め、その期限内に処理を終了するよう、優先度を動的に変更して期限が早いタスクから順に実行 利点：高負荷な環境でも全てのデッドラインを守ることができる 欠点：過負荷状態のとき、デッドラインを守れなくなるプロセスを予測できない • プリエンプティブ • リアルタイム処理でよく用いられる

優先度は、p.28にもあるようにOSが動的に設定することもあります。たとえば、入出力を多く行うプロセスには高い優先度を与えるということが考えられます。

ラウンドロビンは、FCFSや優先度順の欠点を解決します。
タイムスライスが短いとタイマ割込みやスケジューリングの管理オーバーヘッドが大きくなり、逆に長いとラウンドロビンの効果が期待できなくなります。

締切順スケジューリングについては次ページを参照

リアルタイムスケジューリングについて

- リアルタイム（実時間）とは
  処理を設定された時間通りに動作させる。締切りを守る
  - ハードリアルタイム
    締切り（デッドライン）内に終了しなかった時、大きな損害を与え、処理の続行が無意味になる
    - 航空機や自動車の制御など
  - ソフトリアルタイム
    締切りが守れなくても、システム全体に致命的なダメージを与えることはなく、その処理を行うことの意味がある（徐々に落ちていくなど）
    - 音声や画像の再生、銀行ATM処理など
  - リアルタイム処理向けのスケジューリングアルゴリズム
    - 締切り順スケジューリング（Earliest Deadline First）
    - レートモノトニックスケジューリング（Rate Monotonic Scheduling）
      … 周期の短いものにより高い優先度を与える

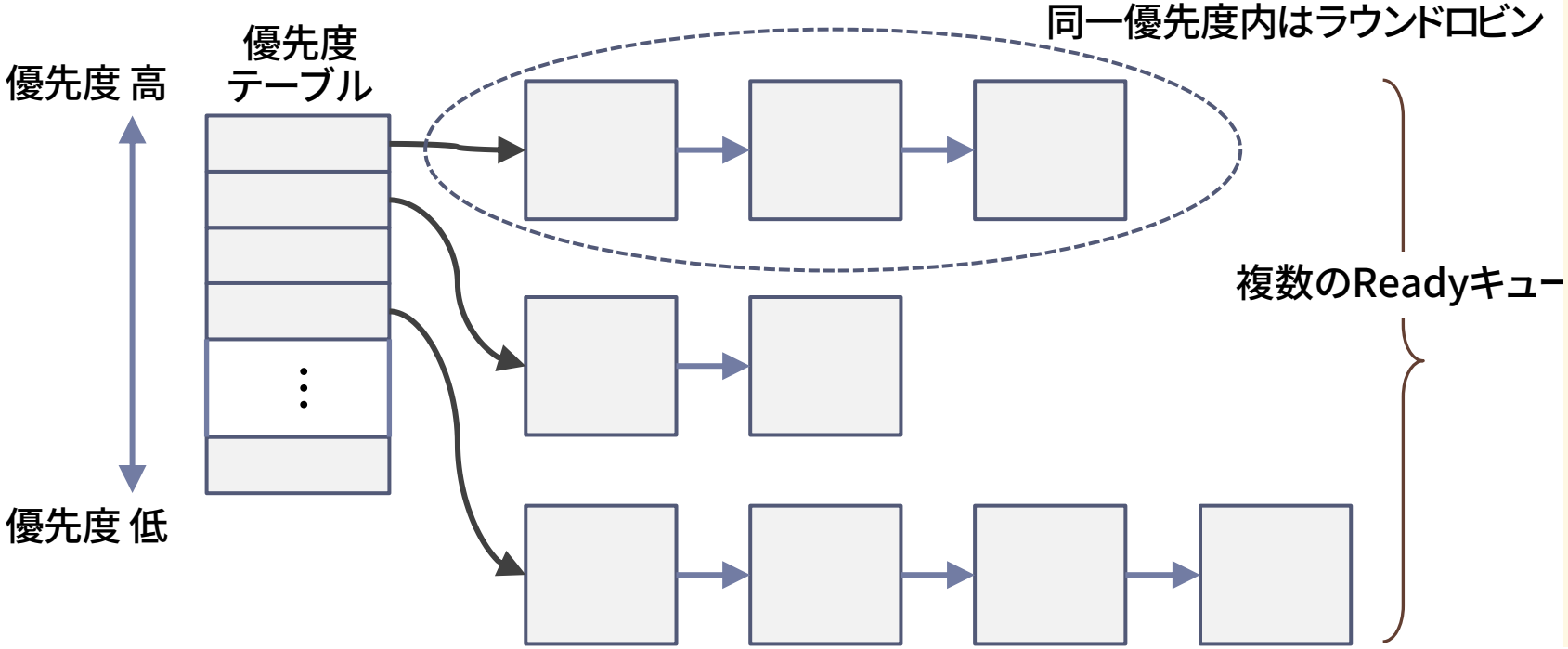
処理時間での締切りを守ることが重要なソフトウェアがあります。自動車のエンジン制御やエアバッグ制御、動画の再生といったものです。たとえば動画の再生では、1/30秒などごとに1コマの画像を出すことになりますが、この1/30秒が締切りに相当します。

そのようなリアルタイム処理に向けたスケジューリングがあります。前ページのデッドラインスケジューリングや、締切り順、レートモノトニックなどがあります。

優先度とラウンドロビンの組み合わせ

- 多重レベルフィードバックキュー
  多くのOSでは、ラウンドロビンと優先度順を組み合わせ、動的に優先度を変化させる方式を採用
  - 高優先度のプロセスは一定時間時間実行したら、優先度を下げる
  - 低優先度のプロセスは一定時間実行されない場合は優先度を上げる

ダイナミックディスパッチング



これまで挙げたスケジューリングは単純でオーバーヘッドが少ないものですが、いずれも欠点があります。たとえば、ラウンドロビンではシステム管理のプロセスのような高優先度のプロセスを設定できないとか、優先度順では低優先度のプロセスの実行機会が少なくなってしまう、などです。
そこで、多くのOSでは、この2つを組み合わせて動的に優先度を変化させる方式をとることが多く採用されます。優先度ごとにラウンドロビンのキューを持ち、実行に従って優先度を変更して行くというものです。

教科書との対応

- スレッドについては、第9回「プロセス管理について(2)」で扱います

第４回の課題

- 1つのCPUと1つの出力装置で構成されるシステムで次の3つのプロセスP1、P2、P3を実行するとき、(a)到着順、(b)処理時間順のアルゴリズムでスケジュールした場合の
  - P1、P2、P3、それぞれの実行完了までに要する時間
  - CPU、および出力の利用効率(実際にCPUや出力が利用された時間の全体の時間に占める割合)

  を示せ
  - また、それぞれのスケジューリングアルゴリズムの特徴はこのスジューリング結果でどのように表されているかを述べよ

  以下の前提を置く:

  ・それぞれの処理時間は右表のとおりとする

  ・P1、P2、P3 の順にシステムに到着するが、システムの動作開始時点ではどれもすでに到着しているとする

  ・スケジューリングの処理自体に要する時間は無視するものとする

	CPU処理時間	出力処理時間
P1	35秒	10秒
P2	20秒	20秒
P3	5秒	25秒

今回の課題です。実際にスケジューリングを行うものです。
クラスウェブのレポートで提出してください。

提出期限は、5/11の午前中とします。

事後学習・事前学習

- 今回の講義資料に基づいて内容を振り返り、教科書などの該当箇所を読む
- 教科書第9章（9.1、9.2）に目を通す

今回の講義内容の振り返りと次回の準備をお願いします。