1. 講義資料：CPUスケジューリングの基礎

第1章 スケジューリングの概要

- **1-1. なぜスケジューリングが必要か？**
  - CPU時間は、複数のプロセスやタスクが共有して利用する**貴重なリソース**です。
  - どのプロセスに、いつ、どれくらいの時間CPUを割り当てるかを決定する仕組みが**スケジューリング**であり、システムの応答性や効率を左右する重要な機能です。
- **1-2. プロセスの挙動**
  - プロセスは、計算処理とI/O処理を繰り返すという特徴があります。この特性から、プロセスは2種類に大別されます。
    - **CPUバウンド (Compute-bound)**: 計算処理が多く、CPU時間を長く消費するプロセス。
    - **I/Oバウンド (I/O-bound)**: I/O待ちが多く、CPU時間は少ししか使わないプロセス。
  - CPUが高速化するにつれて、プロセスはI/Oバウンドになる傾向があります。
- **1-3. スケジューリングのタイミングと方式**
  - スケジューリングは、プロセスの生成・終了、I/Oによるブロック、割り込み発生時などに行われます。
  - スケジューリングの方式には、大きく分けて2つのタイプがあります。
    - **ノンプレンプティブ（横取り無し）**: 一度CPUを割り当てられたプロセスは、自発的にCPUを解放する（ブロックする、または終了する）まで実行を続けます。
    - **プレンプティブ（横取り有り）**: プロセスは一定時間（**クォンタム**）だけ実行を許され、時間が来るとクロック割り込みによって強制的にCPUを奪われ、スケジューラが次のプロセスを選びます。

第2章 スケジューリングの目標

- **2-1. システム環境による分類**
  - スケジューリングアルゴリズムは、システムの利用環境に応じて設計されます。
    - **バッチシステム**: 大量のジョブをまとめて処理する環境。
    - **対話型システム**: ユーザーとコンピュータがリアルタイムに対話する環境。
    - **リアルタイムシステム**: 厳密な時間制約（デッドライン）を守る必要がある環境。
- **2-2. アルゴリズムの評価指標と目標**
  - **全てのシステムに共通する目標**:
    - **公平性 (Fairness)**: 各プロセスに公平にCPUを割り当てる。
    - **システムのバランス**: システムの各部を常に最大限稼働させる。
  - **システムごとの目標**:
    - **バッチシステム**:
      - **スループット**: 単位時間あたりのジョブ処理数を最大化する。
      - **ターンアラウンド時間**: ジョブの投入から完了までの時間を最小化する。
    - **対話型システム**:
      - **応答時間**: 要求から最初の応答が得られるまでの時間を短くする。
    - **リアルタイムシステム**:
      - **デッドラインの遵守**: データ損失などを避けるため、時間的制約を必ず守る。

第3章 具体的なスケジューリングアルゴリズム

- **3-1. バッチシステム向けアルゴリズム**
  - **到着順サービス (First-Come, First-Served)**: CPUを要求した順に割り当てる最も単純なノンプレンプティブ方式です。公平ですが、CPUバウンドなプロセスが後続のI/Oバウンドなプロセスを長時間待たせてしまうという欠点があります。
  - **最短ジョブ優先 (Shortest Job First)**: 実行時間が最も短いと予測されるジョブから順に実行します。平均ターンアラウンド時間を短縮できる最適なノンプレンプティブ方式ですが、事前に実行時間が必要です。

    - **最小残り時間優先 (Shortest Remaining Time Next)**: 上記のプレンプティブ版です。新しいジョブが到着した際、現在実行中のプロセスの残り時間よりも新ジョブの実行時間の方が短ければ、CPUを横取りして新しいジョブを実行します。
- **3-2. 対話型システム向けアルゴリズム**
    - **ラウンドロビン (Round-Robin)**: 最も代表的なプレンプティブ方式です。全てのプロセスを一つのキューで管理し、各プロセスに「クォンタム」と呼ばれる一定時間ずつCPUを割り当てます。
        - クォンタムが**短すぎる**と、プロセス切り替えのオーバーヘッドが大きくなります。
        - クォンタムが**長すぎる**と、応答時間が悪化します。
    - **優先度スケジューリング (Priority Scheduling)**: 各プロセスに優先度を割り当て、最も優先度の高いプロセスから実行します。優先度の低いプロセスが全く実行されない「飢餓状態（starvation）」に注意が必要です。優先度は、CPU使用量やI/Oの頻度に応じて動的に変更されることもあります。
    - **宝くじスケジューリング (Lottery Scheduling)**: プロセスに「くじ」を持たせ、くじ引きによって実行するプロセスを確率的に決定します。重要なプロセスには多くのくじを与えることで、CPU時間を比例配分できます。
    - **公平共有スケジューリング (Fair-Share Scheduling)**: プロセス単位ではなく、**ユーザー単位**でCPU時間の公平性を考慮します。これにより、一人のユーザーが多数のプロセスを起動しても、他のユーザーのCPU時間を不当に奪うことを防ぎます。

第4章 その他のトピック

- **4-1. リアルタイムシステム**
    - **ハードリアルタイム**（デッドライン遵守が絶対）と**ソフトリアルタイム**（多少の遅延は許容）に分類されます。
    - デッドラインが最も近いプロセスを優先する**EDF (Earliest Deadline First)** など、専用のアルゴリズムが用いられます。
- **4-2. ポリシーとメカニズムの分離**
    - OS設計における考え方の一つです。
    - OSは、優先度キューのようなスケジューリングの**メカニズム（仕組み）だけを提供**し、どのプロセスをどの優先度にするかという**ポリシー（方針）は、ユーザープロセス側で決定**できるように分離するアプローチです。

---

2. 重要用語の抽出と説明

- **スケジューリング (Scheduling)**
    - **説明**: どのプロセスに、いつ、どれくらいの時間CPUを割り当てるかを決定するOSの機能です。
- **CPUバウンド / I/Oバウンド (Compute-bound / I/O-bound)**
    - **説明**: **CPUバウンド**は計算処理が中心のプロセスで、**I/Oバウンド**はディスクアクセスなどのI/O待ちが多いプロセスを指します。このプロセスの性質の違いは、スケジューリングの性能に大きく影響します。
- **プレンプティブ / ノンプレンプティブ (Preemptive / Non-preemptive)**
    - **説明**: **ノンプレンプティブ**はプロセスが自発的にCPUを離すまで実行を続ける方式です。一方、**プレンプティブ**はOSが一定時間（クォンタム）でプロセスからCPUを強制的に奪い、他のプロセスに切り替えることができる方式です。
- **クォンタム (Quantum)**
    - **説明**: プレンプティブなスケジューリングにおいて、一つのプロセスが連続してCPUを使用できる時間の最大単位です。この値の設計は、システムのオーバーヘッドと応答性能のトレードオフになります。
- **スループット (Throughput)**
    - **説明**: システムが単位時間あたりに完了できる仕事（ジョブやプロセス）の量です。主にバッチシステムの性能指標として重要視されます。
- **ターンアラウンド時間 (Turnaround Time)**
    - **説明**: プロセス（ジョブ）がシステムに投入されてから、その実行が完全に完了するまでの総時間です。

バッチシステムで最小化が目指されます。
- **応答時間 (Response Time)**
  - **説明**: ユーザーがコマンドを入力するなど要求を出してから、システムが最初の応答を返すまでの時間です。対話型システムの使いやすさを測る上で最も重要な指標です。

---

3. 追加で学習すべき項目

提供された資料は、スケジューリングの基本的な概念と古典的なアルゴリズムを網羅していますが、さらに理解を深めるために以下の項目について学習することをお勧めします。

- **現代のOSにおけるスケジューラの実装**
  - **説明**: 資料で紹介されているアルゴリズム（例: CTSS, XDS940）は概念を理解する上で重要ですが、古典的なものです。現在主流のOS、例えば**LinuxのCFS (Completely Fair Scheduler)** や**Windowsのスケジューラ**が、資料にある公平性 や優先度 といった概念をどのように、より洗練された形で実装しているかを学ぶと、理論と現実の繋がりがより明確になります。
  - **学習のポイント**: CFSがどのように「公平性」を実現しているのか（仮想実行時間の概念など）、また、Windowsがどのように対話型タスクの応答性を高めているのか（動的な優先度ブーストなど）を調べると良いでしょう。
- **マルチプロセッサ・マルチコア環境のスケジューリング**
  - **説明**: 資料は主にCPUが1つである（シングルプロセッサ）環境を前提としています。しかし現代のコンピュータは複数のCPUコアを持つのが一般的です。複数のコアにプロセスやスレッドをどう効率的に割り当てるかという問題は、シングルプロセッサ環境にはない複雑さを持っています。
  - **学習のポイント**: **負荷分散（Load Balancing）**、特定のコアにプロセスを固定する**プロセッサアフィニティ（Processor Affinity）**、さらにはコアのアーキテクチャ（例: big.LITTLE）を考慮したスケジューリングなど、マルチコア特有の課題と解決策について学習することをお勧めします。
- **スレッドスケジューリング**
  - **説明**: 資料の目次には「スレッドスケジューリング」という項目がありますが、その詳細な説明は含まれていません。プロセス内の複数のスレッドをどのようにスケジューリングするかは、プロセス自体のスケジューリングとは異なる考慮点があります。
  - **学習のポイント**: OSカーネルがスレッドを認識してスケジューリングする**カーネルレベルスレッド**と、ユーザー空間のライブラリが管理する**ユーザーレベルスレッド**では、スケジューリングの主体や効率が異なります。これらの違いや、両者を組み合わせたハイブリッドモデルがどのように動作するのかを学ぶことで、並行・並列プログラミングのパフォーマンスに関する理解が深まります。