26. 組み込みアプリケーション開発に関する知識Ⅰ

1. 科目の概要

組み込みシステムで動作するアプリケーション開発について、アプリケーションの基本的な設計と特徴を解説し、さらに資源管理や資源の共有、割り込みに関する処理などについての具体的な実装について説明する。

2. 習得ポイント

本科目の学習により習得することが期待されるポイントは以下の通り。

習得ポイント	説　明	シラバスの対応コマ
1-26-1. コンテキストスイッチの仕組み	複数の処理(タスク)を実行するコンテキストスイッチの仕組みを解説する。OSが処理の切り替えを管理するプリエンプション方式の実装方法を説明し、割り込みによるコンテキストスイッチにも触れる。	1
1-26-2. 非同期処理と同期処理の実装パターンと特徴	非同期/同期の概念について触れ、その実装パターンと設計について内容と特徴を説明する。また実行モードの種類や割り込みの優先順位といった具体的な事項について説明する。	2
1-26-3. タスク優先順位制御とカーネルによる時間管理方法	タスクの優先度とその制御仕様について、実装パターンと設計内容、特徴、手順を説明する。実際のタスク制御を行うカーネルが時間制御方法としてどのような管理手法を用いているかについても触れる。	3
1-26-4. システムリソースの配分の仕組み(セマフォ、キュー、等)	組み込みシステムにおいては、システムリソースが限定されているため様々な工夫が必要になる。各アプリケーションが限られたシステム資源を効率的に共有する方法について、セマフォ、キュー、といった具体的な実装方法を踏まえて、その内容と特徴を説明する。	4
1-26-5. システムリソースの共有の仕組み(共有ファイル、デッドロック回避、等)	複数の組み込みアプリケーション同士でシステムリソースを共有する方法を、共有エリア・共有ファイル、カーネルによる各種のサービス、デッドロックの回避、割り込みディスパッチなどの具体的な実装を挙げて説明する。	5
1-26-6. CPUによるメモリ資源の管理方法	システムリソースのうち、とくにCPUが介在するメモリ空間の管理方法に焦点を当て、ベースアドレス方式、セグメント方式、ページング方式、バンクメモリ方式等、その具体的な実装方法を説明する。	6
1-26-7. メモリの共有制御とプログラムパーティション	複数のアプリケーションがメモリを共有するための制御方式として、固定長方式や可変長方式があることを示し、それぞれの特徴、利点と欠点について説明する。さらにプログラムパーティションの管理、ガベージコレクションといった話題にも言及する。	6
1-26-8. プログラム資源の管理、複数タスクによる同時利用	プログラム資源の管理、有効活用の方式、実装方法について解説する。また複数タスクの同時利用方法として、処理待ちキュー、タスクの優先順位、入出力装置による待ち時間の勘案といったタスク管理も説明する。	7
1-26-9. 入出力資源の管理方法、同時利用方法	実デバイスを操作することが多い組み込みシステムにおいては、とくに入出力装置に対する待ち時間を有効に利用することが求められる。ここでは、入出力待ち時間を有効活用するための方法と、各種の制約、特徴、実装上の留意点などについて述べる。	8
1-26-10. リソースに対する優先順位の設定、割り込みの優先順位	組み込みアプリケーションの非同期性について解説し、リソースの排他的利用、リソース利用に対する優先順位設定、割り込みの優先順位、デバイスドライバ処理の優先順位など、各種の順位決定処理を説明する。	8

【学習ガイダンスの使い方】

1. 「習得ポイント」により、当該科目で習得することが期待される概念・知識の全体像を把握する。
2. 「シラバス」、「IT 知識体系との対応関係」、「OSS モデルカリキュラム固有知識」をもとに、必要に応じて、従来の IT 教育プログラム等との相違を把握した上で、具体的な講義計画を考案する。
3. 習得ポイント毎の「学習の要点」と「解説」を参考にして、講義で使用する教材等を準備する。

3. IT 知識体系との対応関係

「26. 組み込みアプリケーション開発に関する知識Ⅰ」と IT 知識体系との対応関係は以下の通り。

科目名	基本レベル（Ⅰ）								応用レベル（Ⅱ）						
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
26. 組み込みアプリケーション開発に関する知識	<タスクとコンテキスト>	<非同期と同期の設計仕様>	<タスクの優先度とその制御仕様>	<組み込みアプリケーション間の資源配分技術>	<組み込みアプリケーション間のリソースの共有技術>	<リソース有効活用のアーキテクチャ>	<プログラム資源の有効活用技術>	<入出力待ちリソースの有効利用>	<入出力資源管理>	<J2MEの仕様>	<VMの概要のその活用>	<高信頼性の実装>	<組み込みアプリケーションの実装事例研究>	<クリティカルなマイコンアプリケーションを活用した設計ケースワーク>	<組み込みアプリケーションソフトウェアの実装>

[シラバス：http://www.ipa.go.jp/software/open/ossc/download/Model_Curriculum_05_26.pdf]

<IT 知識体系上の関連部分>

分野		科目名	1	2	3	4	5	6	7	8	9	10	11	12	13
組織間連携事項と情報システム	1	IT-IAS 情報保証と情報セキュリティ	IT-IAS1. 基礎的な問題	IT-IAS2. 情報セキュリティの仕組み（対策）	IT-IAS3. 運用上の問題	IT-IAS4. ポリシー	IT-IAS5. 攻撃	IT-IAS6. 情報セキュリティ分野	IT-IAS7. フォレンジック（情報証拠）	IT-IAS8. 情報の状態	IT-IAS9. 情報セキュリティサービス	IT-IAS10. 脅威分析モデル	IT-IAS11. 脆弱性		
	2	IT-SP 社会的な観点とプロフェッショナルとしての課題	IT-SP1. プロフェッショナルとしてのコミュニケーション	IT-SP2. コンピュータの歴史	IT-SP3. コンピュータを取り巻く社会環境	IT-SP4. チームワーク	IT-SP5. 知的財産権	IT-SP6. コンピュータの法的問題	IT-SP7. 組織の中のIT	IT-SP8. プロフェッショナルとしての倫理的な問題と責任	IT-SP9. プライバシーと個人の自由				
応用技術	3	IT-IM 情報管理	IT-IM1. 情報管理の概念と基礎	IT-IM2. データベース問合わせ言語	IT-IM3. データアーキテクチャ	IT-IM4. データモデリングとデータベース設計	IT-IM5. データと情報の管理	IT-IM6. データベースの応用分野							
	4	IT-WS Webシステムとその技術	IT-WS1. Web技術	IT-WS2. 情報アーキテクチャ	IT-WS3. デジタルメディア	IT-WS4. Web開発	IT-WS5. 脆弱性	IT-WS6. ソーシャルソフトウェア							
ソフトウェアの方法と技術	5	IT-PF プログラミング基礎	IT-PF1. 基本データ構造	IT-PF2. プログラミングの基本的構成要素	IT-PF3. オブジェクト指向プログラミング	IT-PF4. アルゴリズムと問題解決	IT-PF5. イベント駆動プログラミング	IT-PF6. 再帰							
	6	IT-IPT 技術を統合するためのプログラミング	IT-IPT1. システム間通信	IT-IPT2. データ割り当てと交換	IT-IPT3. 統合的コーディング	IT-IPT4. スクリプティング手法	IT-IPT5. ソフトウェアセキュリティの実現	IT-IPT6. 種々の問題	IT-IPT7. プログラミング言語の概要						
	7	CE-SWE ソフトウェア工学	CE-SWE0. 歴史と概要	CE-SWE1. ソフトウェアプロセス	CE-SWE2. ソフトウェアの要求と仕様	CE-SWE3. ソフトウェアの設計	CE-SWE4. ソフトウェアのテストと検証	CE-SWE5. ソフトウェアの保守	CE-SWE6. ソフトウェア開発・保守ツールと環境	CE-SWE7. ソフトウェアプロジェクト管理	CE-SWE8. 言語翻訳	CE-SWE9. ソフトウェアのフォールトトレランス	CE-SWE10. ソフトウェアの構成管理	CE-SWE11. ソフトェアの標準化	
	8	IT-SIA システムインテグレーションとアーキテクチャ	IT-SIA1. 要求仕様	IT-SIA2. 調達/手配	IT-SIA3. インテグレーション	IT-SIA4. プロジェクト管理	IT-SIA5. テストと品質保証	IT-SIA6. 組織の特性	IT-SIA7. アーキテクチャ						
システム基盤	9	IT-NET ネットワーク	IT-NET1. ネットワークの基礎	IT-NET2. ルーティングとスイッチング	IT-NET3. 物理層	IT-NET4. セキュリティ	IT-NET5. アプリケーション分野	IT-NET6. ネットワーク管理							
	10	CE-NWK テレコミュニケーション	CE-NWK0. 歴史と概要	CE-NWK1. 通信ネットワークのアーキテクチャ	CE-NWK2. 通信ネットワークのプロトコル	CE-NWK3. LANとMAN	CE-NWK4. クライアントサーバコンピューティング	CE-NWK5. データのセキュリティと整合性	CE-NWK6. ワイヤレスコンピューティングとモバイルコンピューティング	CE-NWK7. データ通信	CE-NWK8. 組込み機器向けネットワーク	CE-NWK9. 通信技術とネットワーク概要	CE-NWK10. 性能評価	CE-NWK11. ネットワーク管理	CE-NWK12. 圧縮と伸張
			CE-NWK13. クラスタシステム	CE-NWK14. インターネットアプリケーション	CE-NWK15. 次世代インターネット	CE-NWK16. 放送									
	11	IT-PT プラットフォーム技術	IT-PT1. オペレーティングシステム	IT-PT2. アーキテクチャと機構	IT-PT3. コンピュータインフラストラクチャ	IT-PT4. デプロイメントソフトウェア	IT-PT5. ファームウェア	IT-PT6. ハードウェア							
	12	CE-OPS オペレーティングシステム	CE-OPS0. 歴史と概要	CE-OPS1. 並行性 [26-Ⅰ-1, 2, 4, 5]	CE-OPS2. スケジューリングとディスパッチ	CE-OPS3. メモリ管理	CE-OPS4. セキュリティと保護	CE-OPS5. ファイル管理	CE-OPS6. リアルタイムOS [26-Ⅰ-3, 4, 5, 8]	CE-OPS7. OSの概要	CE-OPS8. 設計の原則	CE-OPS9. デバイス管理	CE-OPS10. システム性能評価		
コンピュータアーキテクチャとアーキテクチャ	13	CE-CAO コンピュータのアーキテクチャと構成	CE-CA00. 歴史と概要	CE-CA01. コンピュータアーキテクチャの基礎	CE-CA02. メモリシステムの構成とアーキテクチャ	CE-CA03. インタフェースと通信 [26-Ⅰ-3, 7, 8]	CE-CA04. デバイスサブシステム	CE-CA05. CPUアーキテクチャ	CE-CA06. 性能・コスト評価	CE-CA07. 分散・並列処理 [26-Ⅰ-6]	CE-CA08. コンピュータによる計算	CE-CA09. 性能向上			
複数領域にまたがるもの	14	IT-ITF IT基礎	IT-ITF1. ITの一般的なテーマ	IT-ITF2. 組織の問題	IT-ITF3. ITの歴史	IT-ITF4. IT分野（学科）とそれに関連のある分野（学科）	IT-ITF5. 応用領域	IT-ITF6. IT分野における数学と統計学の活用							
	15	CE-ESY 組込みシステム	CE-ESY0. 歴史と概要	CE-ESY1. 低電力コンピューティング	CE-ESY2. 高信頼性システムの設計	CE-ESY3. 組込み用アーキテクチャ	CE-ESY4. 開発環境	CE-ESY5. ライフサイクル	CE-ESY6. 要件分析	CE-ESY7. 仕様定義	CE-ESY8. 構造設計	CE-ESY9. テスト	CE-ESY10. プロジェクト管理	CE-ESY11. 並行設計（ハードウェア、ソフトウェア	CE-ESY12. 実装
			CE-ESY13. リアルタイムシステム設計 [26-Ⅰ-1, 2, 3]	CE-ESY14. 組込みマイクロコントローラ [26-Ⅰ-6]	CE-ESY15. 組込みプログラム [26-Ⅰ-6]	CE-ESY16. 設計手法	CE-ESY17. ツールによるサポート	CE-ESY18. ネットワーク型組込みシステム	CE-ESY19. インタフェースシステムと混合信号システム	CE-ESY20. センサ技術	CE-ESY21. デバイスドライバ	CE-ESY22. メンテナンス	CE-ESY23. 専門システム	CE-ESY24. 信頼性とフォールトトレランス	

4. OSS モデルカリキュラム固有の知識

OSS モデルカリキュラム固有の知識は特になく、各回の内容は IT 知識体系と共通した組み込みアプリケーションの開発に関する内容を扱う。

科目名	第1回	第2回	第3回	第4回	第5回	第6回	第7回	第8回
26.組み込みアプリケーション開発に関する知識Ⅰ	(1)コンピュータアーキテクチャと(2)カーネルコンテキスト	(1)非同期と同期の設計 (2)タスクの構成資源 (3)実行環境	(1)タスクの優先順位 (2)資源の管理 (3)カーネルによる時間管理	(1)資源配分技術 (3)カーネルの提供する手段を使う	(1)共有エリア・共有ファイル (2)カーネルによるサービス (3)デッドロック (4)割り込みスケジューリング（ディスパッチング）	(1)MPU (2)メモリ資源の管理 (3)メモリ共有制御 (4)プログラムパーティション	(1)プログラム資源の管理 (2)複数タスクの同時使用	(1)入出力待ち時間の有効利用 (2)非同期性 (3)処理の優先順位

（網掛け部分は IT 知識体系で学習できる知識を示し、それ以外は OSS モデルカリキュラム固有の知識を示している）

スキル区分	OSS モデルカリキュラムの科目	レベル
組み込み分野	26　組み込みアプリケーション開発に関する知識 I	基本
習得ポイント	I-26-1. コンテキストスイッチの仕組み	
対応する コースウェア	第1回（タスクとコンテキスト）	

I-26-1. コンテキストスイッチの仕組み

複数の処理（タスク）を実行するコンテキストスイッチの仕組みを解説する。OS が処理の切り替えを管理するプリエンプション方式の実装方法を説明し、割り込みによるコンテキストスイッチにも触れる。

【学習の要点】

＊ 各タスクはそれぞれ固有のコンテキストを持つ。コンテキストとは、実行のスケジューリングごとに必要となる CPU のレジスタ状態であり、プログラムの連続した流れとして見ることができる。

＊ コンテキストスイッチは、スケジューラがあるタスクから別のタスクへ実行を切り替える際に、レジスタの値を退避・復元することにより発生する。

＊ OS のスケジューラが必要に応じて実行中のタスクを強制的に中断し、別のタスクを実行する方式をプリエンプション方式と呼ぶ。

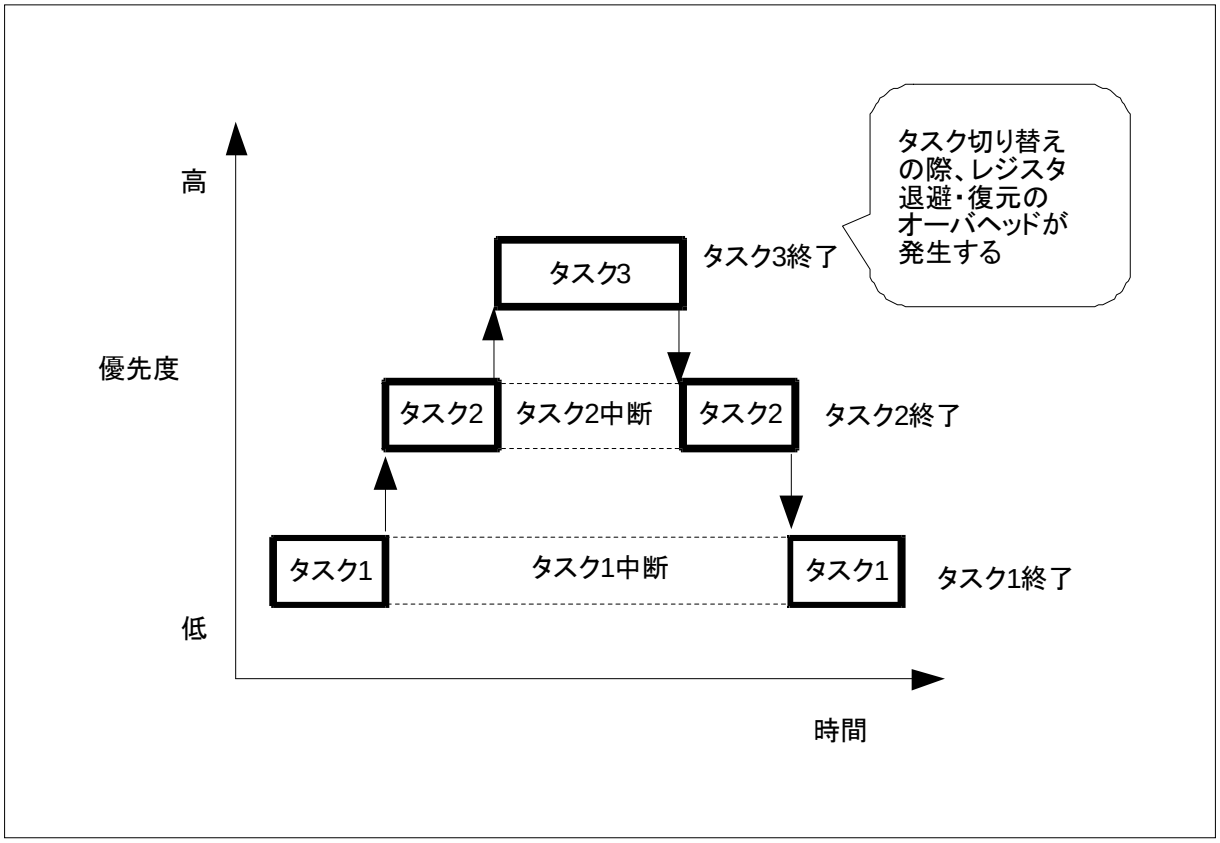

図 I-26-1.　コンテキストスイッチの発生する状況

【解説】

1) コンテキストスイッチング

＊ コンテキストスイッチングとは

- コンテキストとは、実行のスケジューリングごとに必要となる CPU のレジスタ状態であり、プログラムの連続した流れと見られる。
- コンテキストスイッチは、スケジューラがあるタスクから別のタスクへ実行を切り替える際に行うレジスタの値の退避・復元により発生する。
- OS のスケジューラが必要に応じて実行中のタスクを強制的に中断し、別のタスクを実行する方式をプリエンプション方式と呼ぶ。

＊ コンテキストスイッチングの実装仕様

カーネルのスケジューラがタスク 1 からタスク 2 に実行を切り替える時の動作は以下の通り。

- カーネルはタスク 1 のコンテキストをレジスタに退避する
- カーネルはタスク 2 のコンテキストを読み込み、タスク 2 を実行中の状態にする。
- タスク 1 が再び実行状態に復帰する場合、タスク 1 はコンテキストスイッチ直前に実行していた実行状態から再開する。

＊ コンテキストスイッチ時間

- スケジューラがタスク切り替えに要する時間をコンテキストスイッチ時間という。
- 頻繁なコンテキストスイッチを発生させるような設計は、コンテキストスイッチ時間の累積によるパフォーマンスの低下を招く恐れがあり、避けるべきとされる。

2) カーネルコンテキスト

＊ システムコールなどは、通常のアプリケーションが実行されるユーザモードとは異なり、カーネルモード(I-26-2 参照)で実行されるため、カーネルモードの実行に必要なカーネルコンテキストを呼び出す必要がある。

＊ デバイスドライバなどがハードウェア割り込みを行う場合、カーネルの割り込みハンドラを登録する。

＊ 割り込み処理を行っている際のコンテキストを割り込みコンテキストという。

スキル区分	OSS モデルカリキュラムの科目	レベル
組み込み分野	26　組み込みアプリケーション開発に関する知識 I	基本
習得ポイント	I-26-2. 非同期処理と同期処理の実装パターンと特徴	
対応する コースウェア	第 2 回（非同期と同期の設計仕様）	

I-26-2. 非同期処理と同期処理の実装パターンと特徴

非同期/同期の概念について触れ、その実装パターンと設計について内容と特徴を説明する。また実行モードの種類や割り込みの優先順位といった具体的な事項について説明する。

【学習の要点】

* 非同期処理は、あるタスクが実行をしている際に、他のタスクが別の処理を実行できる方式である。同期処理では、あるタスクが実行している間、他のタスクの処理は中断される方式である。
* タスクの情報を保持する管理テーブル（Task Control Block: TCB)から、プログラムへのポインタとコンテキストの情報を取得して、カーネルはタスクを起動する。
* タスクの実行モードにはカーネルモードとユーザモードがある。カーネルモードは全ての命令の実行が許可されるのに対し、ユーザモードでは実行できる命令に制限が加わる。

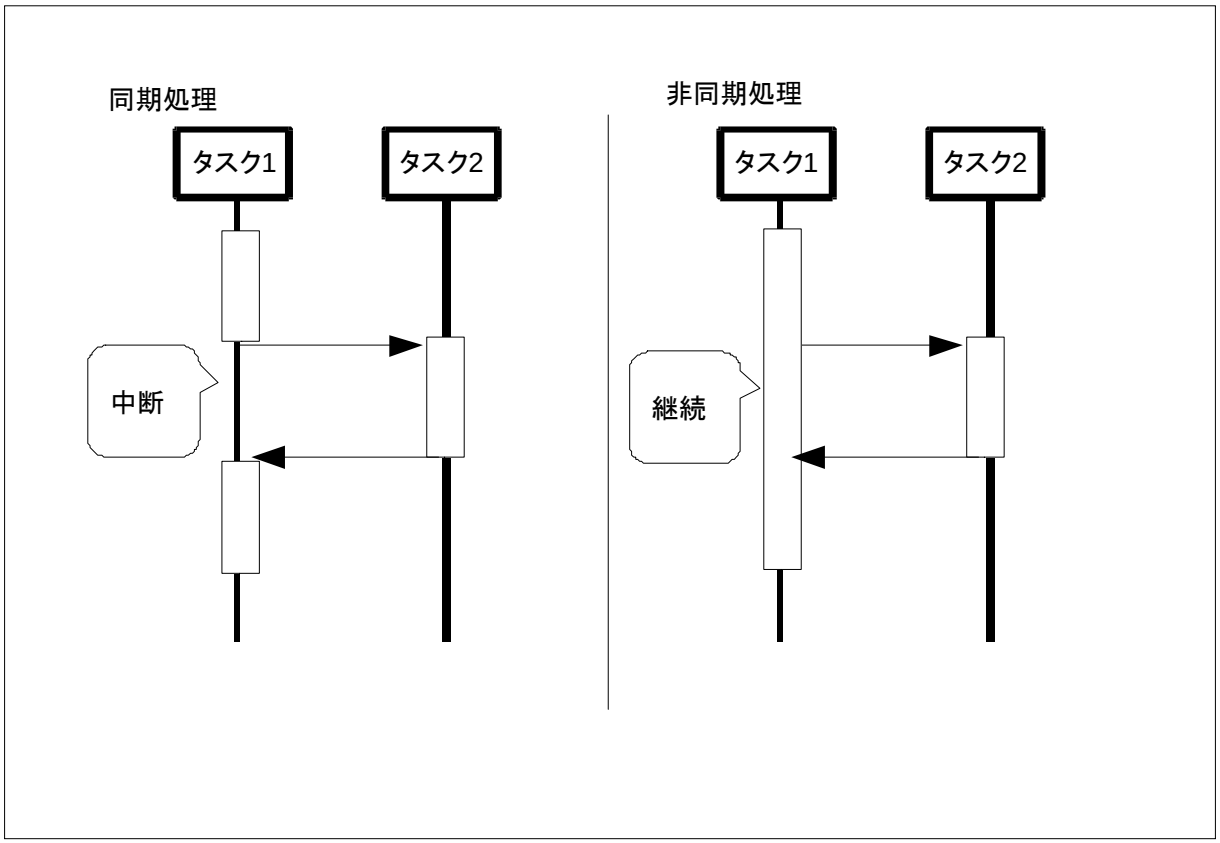

図 I-26-2.　同期処理と非同期処理

【解説】

1) 非同期と同期の設計

非同期処理は、あるタスクが実行をしている際に、他のタスクが別の処理を実行できる方式である。同期処理では、あるタスクが実行している間、他のタスクの処理は中断される方式である。

＊ システム全体のコンテキストの設計

- 非同期処理を入れ、多重にコンテキストを用意することにより処理速度の向上を目指す設計が行われる。

2) タスクの構成資源

＊ タスクの生成

- タスクは生成時に、プログラム部分の他、名前、GUID（一意の ID）、優先度、タスクの情報を保持する管理テーブル（Task Control Block: TCB)、スタックが割り当てられる。

＊ 実行モードと起動方法

- カーネルモード
  OS の実行モードであり、全ての命令の実行が許可される。あらゆるハードウェア資源にアクセスできる。

- ユーザモード
  通常のプログラムの実行モードであり、命令の実行に制限が加わる。ハードウェア資源に制限・監視付きでアクセスできる。

3) 処理の優先順位

＊ 割り込みには優先度がある。優先度の高いタスクの割り込みが優先される。

＊ 割り込みは必要に応じて禁止することができる。割り込みを禁止することを割り込みマスクという。

＊ マスクできない優先度の高いタスクもあり、これを Non-Maskable Interrupt(NMI)という。

スキル区分	OSS モデルカリキュラムの科目	レベル
組み込み分野	26　組み込みアプリケーション開発に関する知識 I	基本 I
習得ポイント	I-26-3. タスク優先順位制御とカーネルによる時間管理方法	
対応する コースウェア	第 3 回（タスクの優先度とその制御仕様）	

I-26-3. タスク優先順位制御とカーネルによる時間管理方法

タスクの優先度とその制御仕様について、実装パターンと設計内容、特徴、手順を説明する。実際のタスク制御を行うカーネルが時間制御方法としてどのような管理手法を用いているかについても触れる。

【学習の要点】

＊ カーネルは優先度を設定してタスクの優先度順実行を制御する。

＊ カーネルによる時間管理には、プロセス実行を時分割で切り替えるタイムスライス方式、割り込み等のイベントを受けてタスクを切り替えるプリエンプティブ方式などがある。

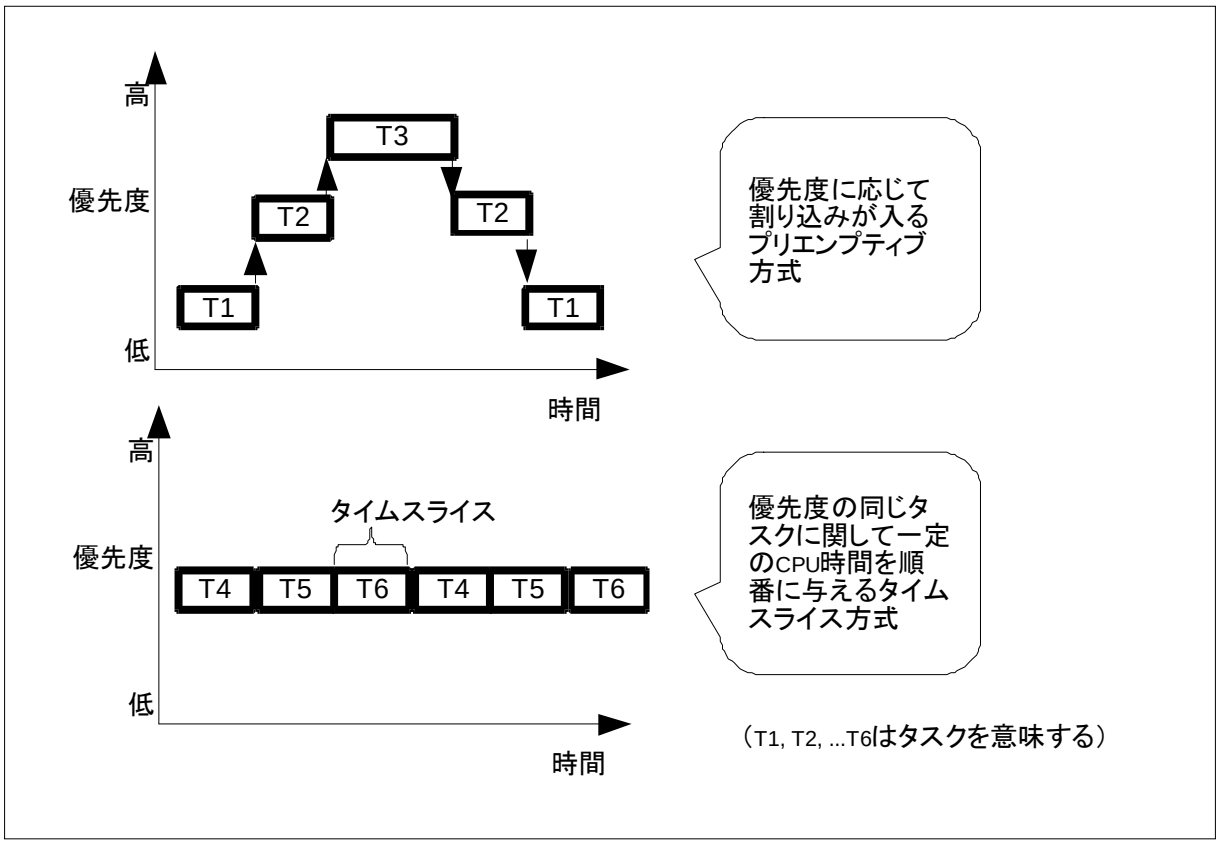

図 I-26-3.　プリエンプティブ方式とタイムスライス方式

【解説】

1) タスクの優先順位

カーネルは優先度を設定してタスクの優先度順実行を制御する。

＊ プログラムのリエントラント性と優先度

- リエントラント(再入可能)とは、プログラムに含まれるあるルーチンの実行中に同じルーチンが呼び出されても正しい結果を返せることを指す。
- C 言語の場合、スタティック変数の更新を行わない関数はリエントラント性が保証される。

＊ 同期／排他制御と優先度

- リエントラントでないルーチンを使う前後で排他制御を用いると、そのルーチンはリエントラントルーチン同様に、複数のプログラムから利用できる。
- 上述の使い方のできるライブラリを特に、スレッドセーフライブラリ(Thread Safe Library)と呼ぶ。

2) カーネルによる時間管理

＊ カーネルによる時間管理には、タスクの実行を時分割で切り替えるタイムスライス方式、割り込み等のイベントを受けてタスクを切り替えるプリエンプティブ方式などがある。

- タイムスライス方式
  複数のタスクを平等に実行させるための方式。タイムスライスという単位時間でタスクを切り替えて実行する方式である。

- プリエンプティブ方式
  外部からの割り込みが発生すると、スケジューリングを行い、優先度に応じて処理を切り替える方式である。

- フィードバック待ち行列
  優先度のある待ち行列にタスクを並べ、順に処理を行う。その際、一定時間内に処理が終わらない場合、優先度を落とす方式である。

スキル区分	OSS モデルカリキュラムの科目	レベル
組み込み分野	26　組み込みアプリケーション開発に関する知識 I	基本
習得ポイント	I-26-4. システムリソースの配分の仕組み(セマフォ、キュー、等)	
対応する コースウェア	第 4 回 （組み込みアプリケーション間の資源配分技術）	

I-26-4. システムリソースの配分の仕組み(セマフォ、キュー、等)

組み込みシステムにおいては、システムリソースが限定されているため様々な工夫が必要になる。各アプリケーションが限られたシステム資源を効率的に共有する方法について、セマフォ、キュー、といった具体的な実装方法を踏まえて、その内容と特徴を説明する。

【学習の要点】

* 組み込みシステムにおいては、CPU リソース、メモリリソース、ネットワークリソース、制御時間といったリソースの配分が重要である。
* 複数のタスクが共有資源にアクセスしようとした際に、整合性を保つために同時アクセスを禁止することを排他制御と呼ぶ。排他制御にはロックやデッカーのアルゴリズム、セマフォなどの方式が使われる。
* カーネルによって制御されているタスク間の排他制御手段としてセマフォがある。セマフォはタスクが共有資源にアクセスするために必要な鍵の役割を果たす。タスクは、セマフォを取得することで共有資源にアクセス可能になる。
* タスク間通信にキューを用いてメッセージを交換することで、同期をする方式もある。

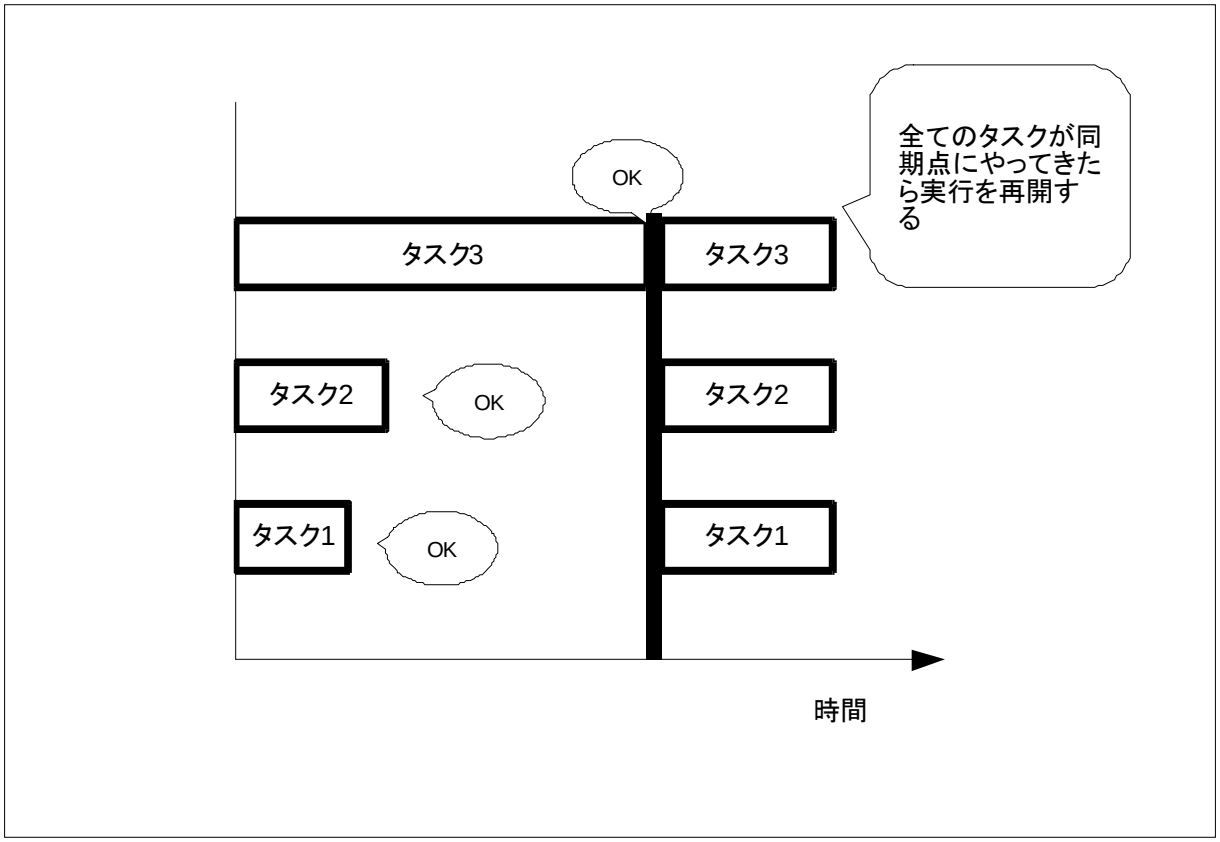

図 I-26-4.　アクティビティ同期の例(バリア同期)

【解説】

1) 資源配分技術

＊ 配分すべき資源には、CPU リソース、メモリリソース、ネットワークリソース、制御時間がある。

＊ クリティカルセクション

- 共有リソースにアクセスするコードの区間であり、プリエンプションが発生することによってデータの整合性などに問題が生じる可能性のある部分を指す。

＊ 同期制御と排他制御

- リソース同期
  共有リソースの整合性を保つために複数タスク間での同期をすること。

- アクティビティ同期
  タスクがアクティビティを他のタスクと同期すること。バリア同期がその例として挙げられる。バリア同期では、複数のタスクがある時間をバリアとして「待ち合わせ」をする。

- スピンロックによる排他制御
  あるタスクがクリティカルセクションに入る直前にフラグをチェックすることで、他のタスクがクリティカルセクションに入っていないことを確認してからクリティカルセクションに入る。他のタスクがクリティカルセクションに入っている場合は、フラグをチェックし続ける。

＊ ハードウェアによる方法

- Test and Set 命令は、スピンロックにおけるフラグのチェックをアトミックに実現する機械語命令である。

＊ ソフトウェアによる方法

- デッカーのアルゴリズム(I-24-9 参照)、ランポートのパン屋のアルゴリズムといった相互排除のアルゴリズムが考案されている。

2) カーネルの提供する手段を使う方法

＊ セマフォ
セマフォはタスクが共有資源にアクセスするために必要な鍵の役割を果たす。セマフォを取得することで共有資源にアクセス可能になる。

- バイナリセマフォ
  セマフォは 0 か 1 の値を持つ。0 ならば共有資源は利用不可、1 ならば利用可能である。

- ジェネラルセマフォ(カウンティングセマフォ)
  カウンタにより、共有資源へのアクセス権の複数回の取得や解放を管理できる。

＊ キューイング

- タスク間通信にキューを用いてメッセージを交換することで、同期をする方式もある。

スキル区分	OSS モデルカリキュラムの科目	レベル
組み込み分野	26　組み込みアプリケーション開発に関する知識 I	基本
習得ポイント	I-26-5. システムリソースの共有の仕組み(共有ファイル、デッドロック回避、等)	
対応する コースウェア	第 5 回 （組み込みアプリケーション間のリソースの共有技術）	

I-26-5. システムリソースの共有の仕組み(共有ファイル、デッドロック回避、等)

複数の組み込みアプリケーション同士でシステムリソースを共有する方法を、共有エリア・共有ファイル、カーネルによる各種のサービス、デッドロックの回避、割り込みディスパッチなどの具体的な実装を挙げて説明する。

【学習の要点】

* アプリケーションプロセス間で共有するデータの参照方式にはシンボル参照、ファイル参照などがある。
* デッドロックを処理する戦略として、デッドロック検出と復旧、デッドロック回避、デッドロック予防がある。

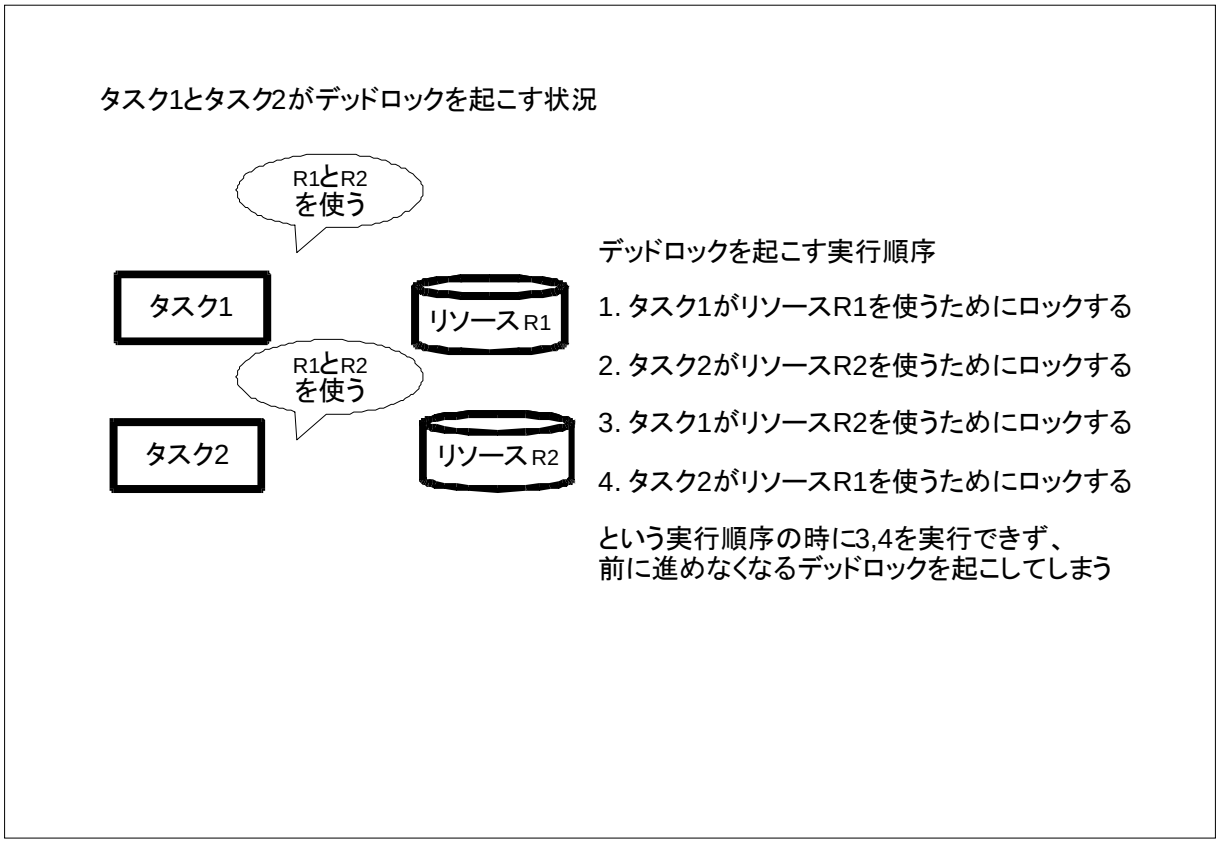

図 I-26-5.　デッドロックを起こす状況の例

【解説】

1) 共有データの参照

組み込みソフトウェアは、モジュール化が進んでいるため、タスク間でデータやライブラリを共有することが多い。

* シンボル参照
  - プロセス空間内の同一領域に、共有データ領域を予約しておく方式。
* ファイル参照
  - メモリ上に擬似的にファイルを構成することで、共有ファイルとしてデータの読み書きを行う方式。

2) カーネルによるサービス

* メッセージキュー
  - 2つのタスク間のデータ通信などに利用されるカーネルオブジェクト。
* パイプ
  - タスク間のデータ交換機構。データは片方のディスクリプタに書き込まれ、もう片方のディスクリプタから読み取られる。Linux などのシェルにおいてパイプは"|"記号で表現され、頻繁に利用される。

3) デッドロック

* デッドロックとは、複数のタスクがリソース要件を満たせずに処理が全く進まなくなってしまう状態である。
* デッドロックを処理する戦略として、デッドロック検出と復旧、デッドロック回避、デッドロック予防がある。これらはリアルタイム OS の機能として提供される。

4) 割り込みスケジューリング（ディスパッチング）

* ディスパッチャはスケジューラの一部であり、実際のコンテキストスイッチ処理を行う。
* システムコールを行う主体がタスクか割り込みサービスルーチン(Interrupt service routines: ISR)かによって処理が異なり、ISR が優先される。

スキル区分	OSS モデルカリキュラムの科目	レベル
組み込み分野	26　組み込みアプリケーション開発に関する知識 I	基本
習得ポイント	I-26-6. CPU によるメモリ資源の管理方法	
対応する コースウェア	第 6 回(リソース有効活用のアーキテクチャ)	

I-26-6. CPU によるメモリ資源の管理方法

システムリソースのうち、とくに CPU が介在するメモリ空間の管理方法に焦点を当て、ベースアドレス方式、セグメント方式、ページング方式、バンクメモリ方式等、その具体的な実装方法を説明する。

【学習の要点】

* 仮想メモリ空間の管理方式には、ベースアドレス方式、セグメント方式、ページング方式がある。
* ベースアドレス方式は、指定されたアドレスにベースアドレスレジスタの値を加えてからメモリにアクセスする方式である。
* セグメント方式は、メモリの複数のセクションに分割し、セクションごとにベースアドレスレジスタを持たせる拡張をした方式である。
* ページング方式は、メモリのページと呼ばれるブロックごと(セクションよりも細かい)にベースアドレスレジスタを持たせる拡張をした方式である。

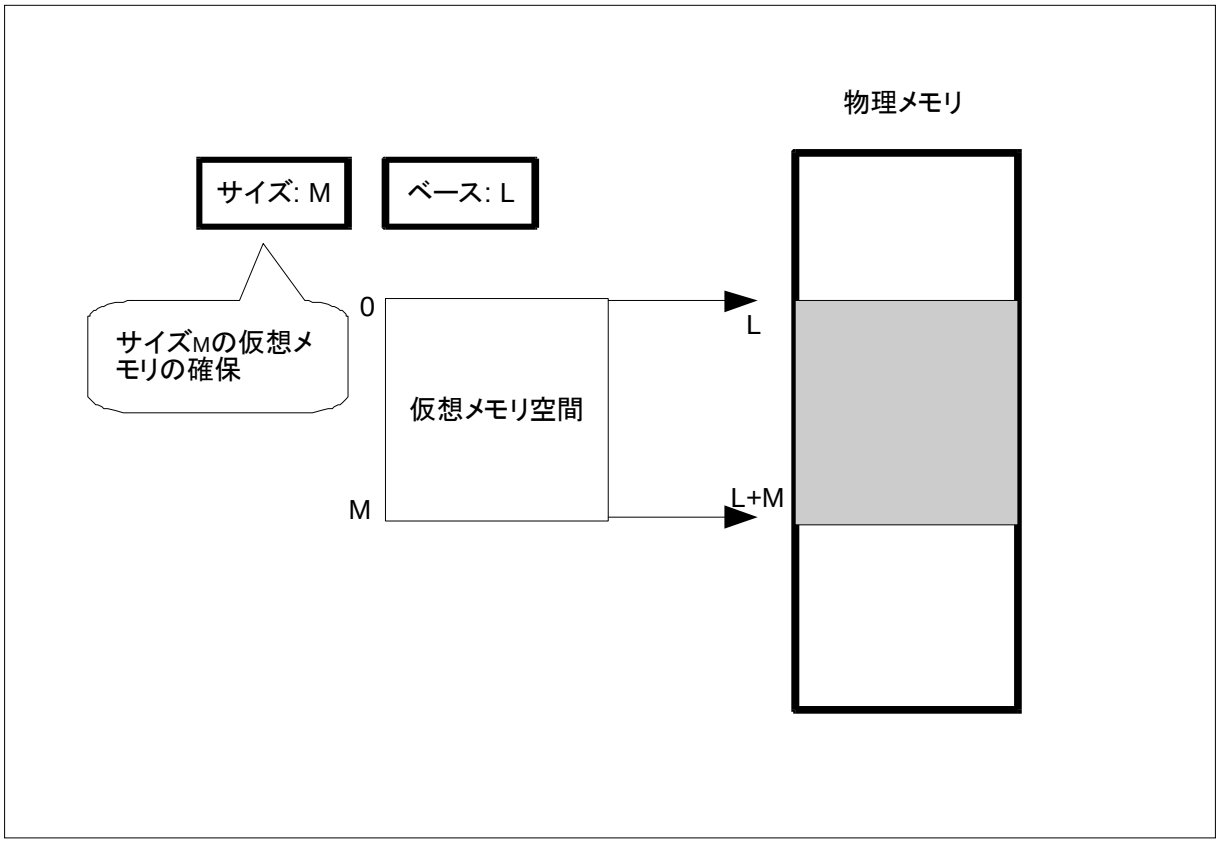

図 I-26-6.　ベースアドレス方式による仮想メモリ管理

【解説】

1）メモリ資源の管理

＊ 仮想メモリはハードディスクの一部をメインメモリの一部として利用するために OS が提供する機能の一つである。仮想メモリ空間の管理を行うことで、物理的なメモリの詳細をプログラマから隠蔽する。仮想メモリの管理方式には以下の方式がある

- ベースアドレス方式
  ベースアドレス方式は、アクセスしようとするアドレスにベースアドレスレジスタの値を加えてからアクセスする方式である。

- セグメント方式
  セグメント方式は、メモリの複数のセクションに分割し、セクションごとにベースアドレスレジスタを持たせる拡張をした方式である。

- ページング方式
  ページング方式は、メモリのページと呼ばれるブロックごと（セクションよりも細かい）にベースアドレスレジスタを持たせる拡張をした方式である。

＊ バンクメモリ

- メモリを複数のバンクに分割し、各バンクに並列にアクセスすることで効率を向上させる。この方式をメモリインタリーブと呼ぶ。

＊ メモリプールの管理

- メモリプールはメモリ確保・解放による処理の煩雑さを軽減するための手法である。
- 一定の確保されたメモリ領域の抽象的な表現であり、プログラムからアクセスできる。

スキル区分	OSS モデルカリキュラムの科目	レベル
組み込み分野	26　組み込みアプリケーション開発に関する知識 I	基本
習得ポイント	I-26-7. メモリの共有制御とプログラムパーティション	
対応する コースウェア	第 6 回 （リソース有効活用のアーキテクチャ）	

I-26-7. メモリの共有制御とプログラムパーティション

複数のアプリケーションがメモリを共有するための制御方式として、固定長方式や可変長方式があることを示し、それぞれの特徴、利点と欠点について説明する。さらにプログラムパーティションの管理、ガベージコレクションといった話題にも言及する。

【学習の要点】

＊ メモリを共有して利用する際には、可変長と固定長の制御方式がある。

＊ メモリの使用時間が経過するにつれ、メモリ利用効率の悪いフラグメンテーションという症状がどちらの方式でも発生する。

＊ ガベージコレクションにより利用されていないメモリの管理をすることで、メモリの利用効率を改善する。

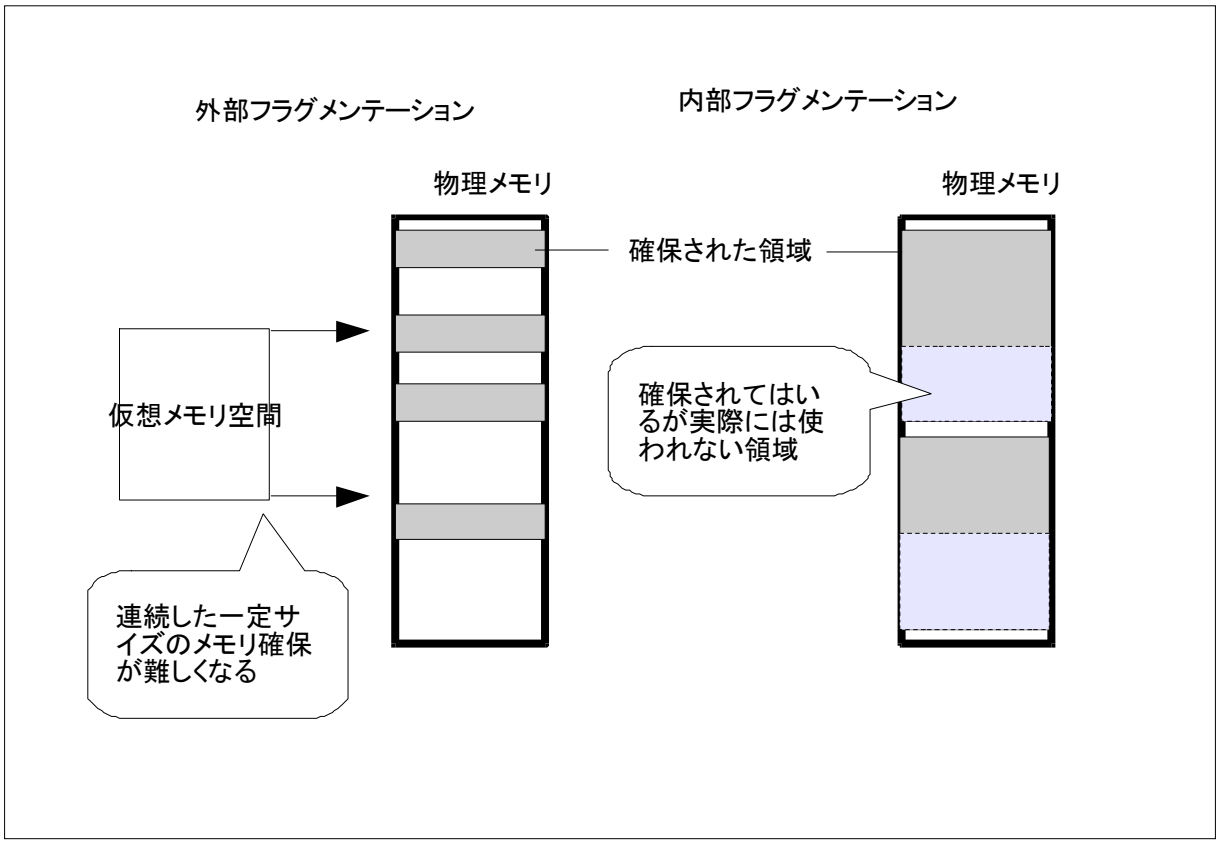

図 I-26-7.　外部フラグメンテーションと内部フラグメンテーション

【解説】

1) メモリ共有制御

＊ メモリを共有して利用する際には、可変長と固定長の制御方式がある。

- 可変長の方式では、メモリを確保するブロックのサイズを変更できる。
- 固定長の方式では、メモリを確保するブロックのサイズは一定である。

＊ どちらの方式にもフラグメンテーションの問題がある

- 固定長で管理する場合には、内部フラグメンテーションが起こる。
  固定長の場合、メモリ領域の固定長サイズと実際に必要なサイズが合わずに未使用の領域を残す。空きブロックが無くなった際に、実際には未使用の領域があっても割り当てができない。

- 可変長で管理する場合には、外部フラグメンテーションが起こる。
  可変長の場合、メモリからのブロックの出し入れが続くと、未使用メモリの多くが小さなブロックに細分化されてしまい、連続したメモリ領域への要求に対応できない。

2) ガベージコレクション

＊ プログラムが使用しなくなった不要なメモリ領域を解放し、利用可能なメモリ領域を増やす技術である。

＊ プログラムミスにより、未解放のメモリ領域が増えていく状況（メモリリーク）によるエラーを防ぐために有効な仕組みである。

＊ ガベージコレクションは、Java, Perl, Python, Ruby などの言語が実装している。

スキル区分	OSS モデルカリキュラムの科目	レベル
組み込み分野	26　組み込みアプリケーション開発に関する知識 I	基本
習得ポイント	I-26-8. プログラム資源の管理、複数タスクによる同時利用	
対応する コースウェア	第 7 回 （プログラム資源の有効活用技術）	

I-26-8. プログラム資源の管理、複数タスクによる同時利用

プログラム資源の管理、有効活用の方式、実装方法について解説する。また複数タスクの同時利用方法として、処理待ちキュー、タスクの優先順位、入出力装置による待ち時間の勘案といったタスク管理も説明する。

【学習の要点】

* 複数のプログラムから同時に呼ばれるルーチンや、メモリに常駐して何度も繰り返し実行されるプログラムは、リユーザブル（再利用可能）、リエントラント（再入可能）、といった性質を満たす必要がある。
* リユーザブルプログラムとは、一度実行した後に再度プログラムを実行しても正しく処理できるプログラムを指す。
* リエントラントプログラムは、プログラムのあるルーチンの実行中に同じルーチンが呼び出されても正しい結果を返せるプログラムを指す。

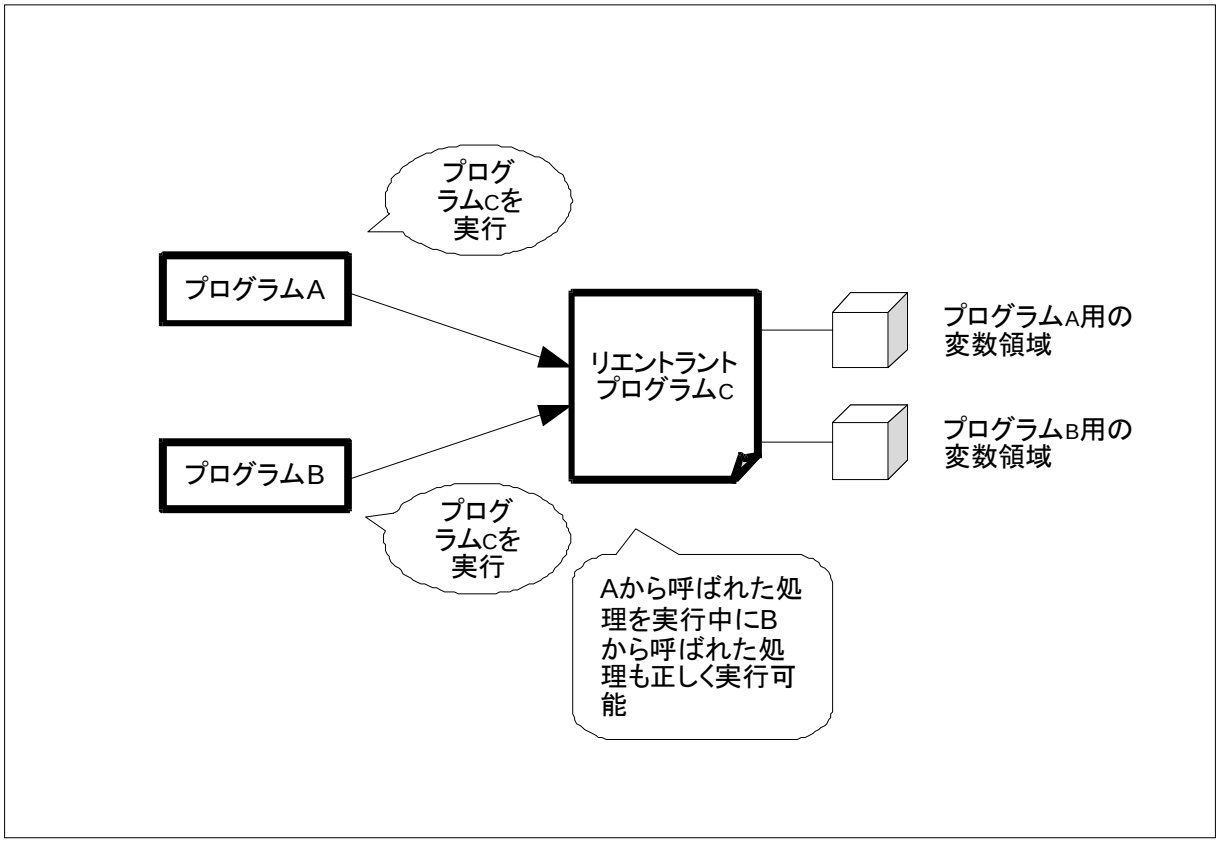

図 I-26-8.　リエントラントプログラム

【解説】

1) プログラム資源の管理

＊ リユーザブルとリエントラント

- リユーザブル(Reusable，再利用可能)プログラムは、一度実行した後に、再度実行しても正確な実行を繰り返す性質をもつプログラムである。
- 組み込みシステムの場合、単一リンクモジュールやメモリ常駐ルーチンにおいてリユーザブルであることが求められる。
- リエントラント(Reentrant，再入)プログラムは、プログラムのあるルーチンの実行中に同じルーチンが実行されても正確な実行を繰り返す性質を持つプログラムである。

＊ ロールインとロールアウト（スワップ）

- OS により必要なプログラムをメモリ上に読み込むことをロールインと呼ぶ。
- 読み込む際に、メモリ領域が不足している場合には、不要なメモリ領域の情報を補助記憶に書きだしてメモリ領域を確保する。この書き出しをロールアウトと呼ぶ。

＊ 動的メモリ管理機能

- プログラムの実行中に動的にメモリの確保・解放を行う仕組み（ガベージコレクション）を提供するプログラミング言語もある(I-26-7 参照)。

＊ ワーキングセット

- ソフトウェアが利用しているメモリのうち、ロールアウトされていない領域をワーキングセットと呼ぶ。

2) 複数タスクの同時使用

＊ 処理待ちキューの作り方

- 処理依頼の到着順
  FIFO と呼ばれる方式である。これは「First In, First Out」の略であり、最初に入ったタスクが最初に処理される方式である。

- タスクの優先順位順
  タスクの優先度の高い順に実行を進める、追い越しを許した処理待ちキューである。タイムスライスという一定時間単位を割り当てるタイムスライス方式で、優先度の高い順にタスクを実行するラウンドロビン方式などがある。

- 入出力装置／他のマイコンとの関係の効率順
  あるタスクの入出力の結果にほかのタスクが依存している場合など、実行効率を考慮した優先度を動的に割り当てる。

スキル区分	OSS モデルカリキュラムの科目	レベル
組み込み分野	26　組み込みアプリケーション開発に関する知識 I	基本
習得ポイント	I-26-9. 入出力資源の管理方法、同時利用方法	
対応する コースウェア	第 8 回（入出力待ちリソースの有効利用）	

I-26-9. 入出力資源の管理方法、同時利用方法

実デバイスを操作することが多い組み込みシステムにおいては、とくに入出力装置に対する待ち時間を有効に利用することが求められる。ここでは、入出力待ち時間を有効活用するための方法と、各種の制約、特徴、実装上の留意点などについて述べる。

【学習の要点】

＊ 入出力ポートは同時に複数のタスクからの要求を受けた場合でも、ひとつのタスクのみの利用に限定される。実行中のタスクが終了するまで他のタスクは待たされる。

＊ 入出力ポートの利用を待ち情報を記憶したキューを入出力キューと呼ぶ。また、セマフォなどにより優先度の管理が行われる。

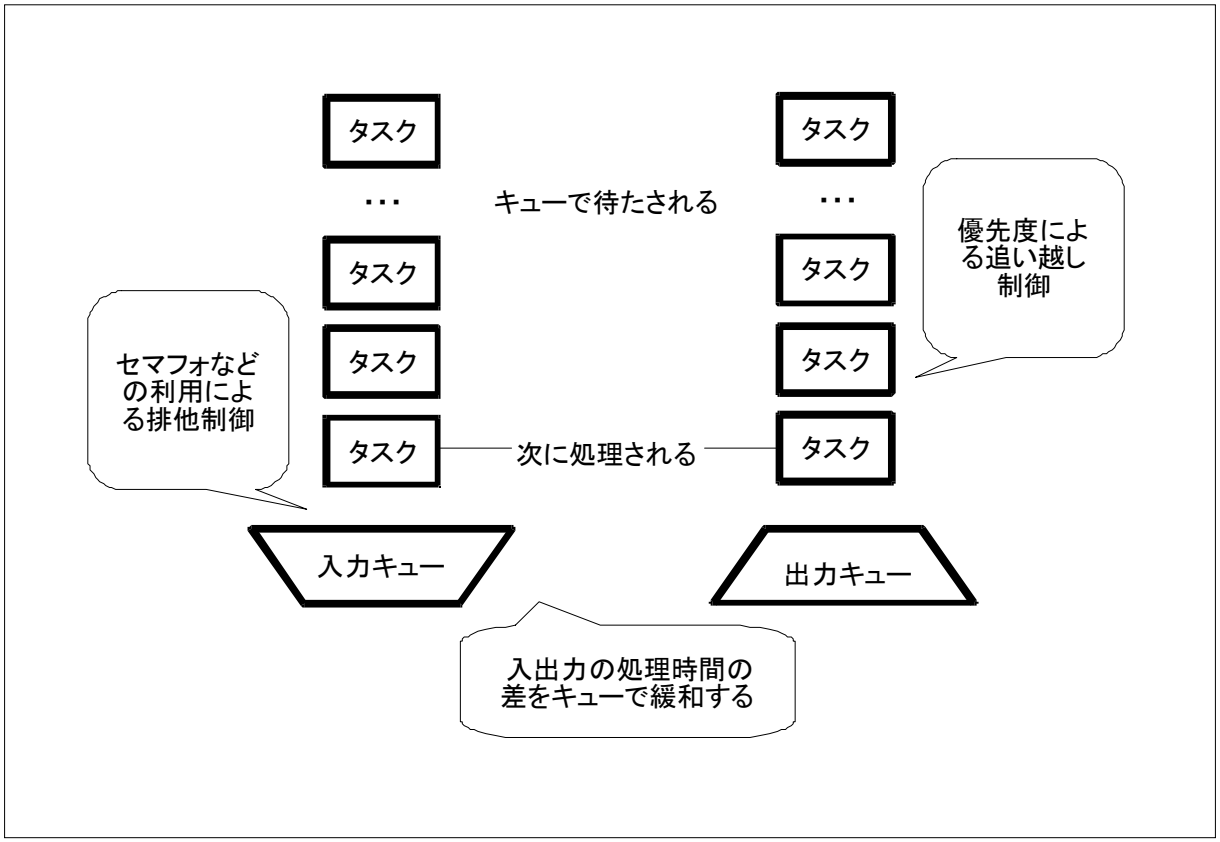

図 I-26-9.　入出力キュー

【解説】

1）入出力待ち時間の有効利用

＊ 入出力制御

- 入出力ポートは同時に複数のタスクからの要求を受けた場合でも、ひとつのタスクのみの利用に限定される。
- 実行中のタスクが終了するまで他のタスクは待たされるため、非効率な状況が発生しうる。入出力資源の効果的な利用が求められる。
- デバイスからの入出力イベントを待つ場合、事前にタスクを生成してイベント待ちにしておくことで、タスク生成のオーバヘッドを軽減することができることが多い。
- ハードディスクなど、入出力データやイベントには入出力要求のキューを作成して処理を待たせる方式が一般に利用される。
- 入力と出力とキューを分ける方式もあり、入出力の速度差をキューで解消することとなる。
- 入出力の速度差に伴う効率を考慮した設計を行うことが、リアルタイム性の要求に応えるために重要である。

＊ 割り込みの利用

- システムで最も優先度の高い割り込みを利用して緊急度の高い処理を実行させる場合もある。

＊ 入出力資源の同時使用方法

- 同時並行処理できるデバイス単位にリエントラント性を保証することにより実現できる。

スキル区分	OSS モデルカリキュラムの科目	レベル
組み込み分野	26　組み込みアプリケーション開発に関する知識 I	基本
習得ポイント	I-26-10. リソースに対する優先順位の設定、割り込みの優先順位	
対応する コースウェア	第 8 回（入出力待ちリソースの有効利用）	

I-26-10. リソースに対する優先順位の設定、割り込みの優先順位

組み込みアプリケーションの非同期性について解説し、リソースの排他的利用、リソース利用に対する優先順位設定、割り込みの優先順位、デバイスドライバ処理の優先順位など、各種の順位決定処理を説明する。

【学習の要点】

* 複数のタスクが同時に入出力に対して要求を出しても、同時に利用できるタスクは一つだけである。シリアライジングを行い、逐次処理を行う必要がある。
* シリアライジングの実現には、セマフォを利用する方式、独自にキューを作成する方式などがある。
* デバイスドライバはアプリケーションインターフェイスと割り込みハンドラのコンテキストから構成される。

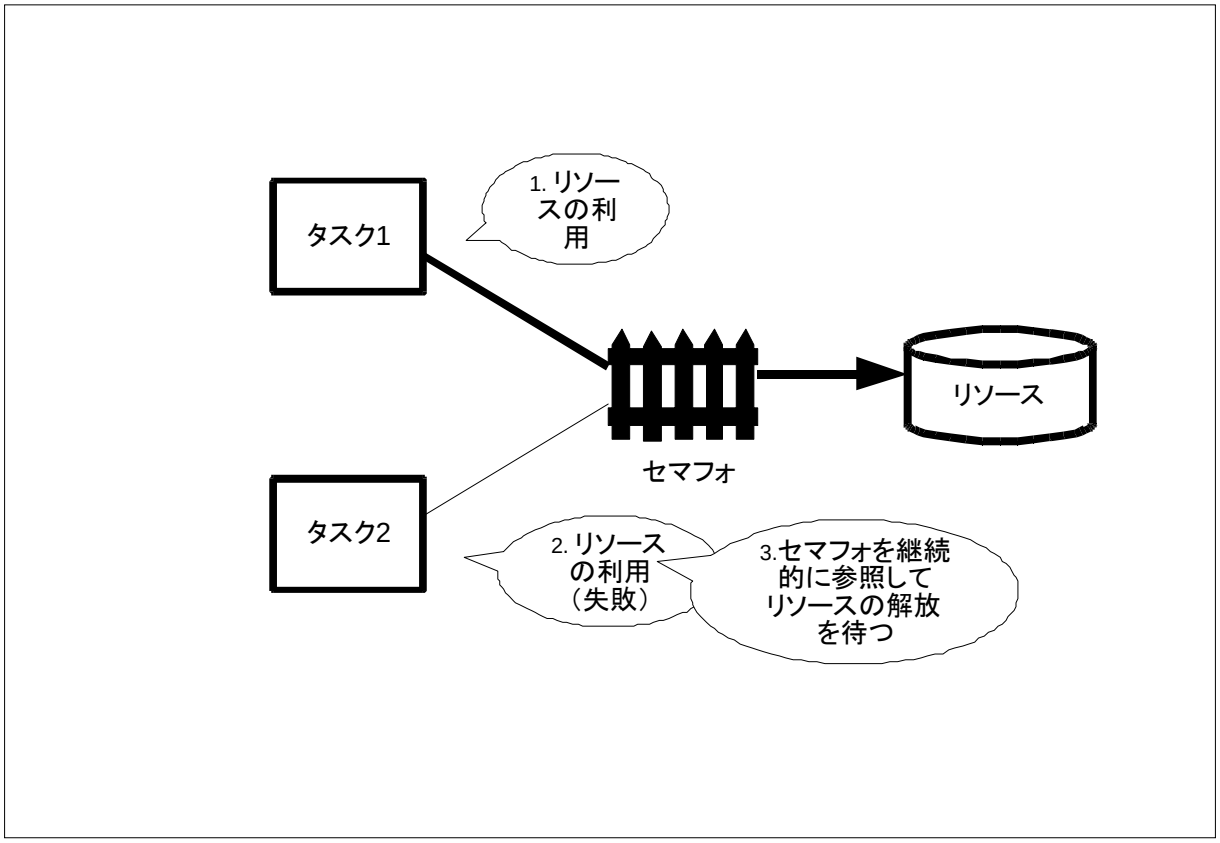

図 I-26-10.　セマフォによる処理の逐次化

【解説】

1) シリアライジング

* 複数のタスクが同時に入出力に対して要求を出しても、同時に利用できるタスクは一つだけである。
* 要求を一つずつ処理することをシリアライジングと呼ぶ。
* 入出力(要求)キュー
  - シリアライジングに際し、入出力の要求を出して待たされている情報を記憶したキューを指す。
* セマフォを利用する方式
  - デバイスドライバ全体をセマフォで占有することにより排他制御を行う。
  - セマフォを利用する場合、タスク管理を行うタスク管理ブロック (Task Control Block : TCB) そのものが入出力キューの役割を果たす。
* 独自にキューを作成する方式
  - システムコールとしてアプリケーションから呼ばれる場合、独自に入出力キューを作成する必要がある。
  - カーネルコンテキストで実行する OS ではこの方式を実装している。

2) アプリケーションとの同期

* 同期入出力
  - 入出力が完了した時点で、入出力に対して要求を出しているタスクの待ち状態を解除する方式。
* 非同期入出力
  - 入出力要求をデバイスドライバが受け付けた後も、デバイスドライバと並行して実行を継続する方式。

3) リアルタイム処理の優先順位

* タイムクリティカルな処理を最速に行う方式は、割り込みハンドラで処理を行う方式である。
* タイムクリティカルな処理が、入力データ待ちや入出力待ちの状況になる場合、デッドロックを避けるため、最高優先度のタスクとして実行するように設計する必要がある。最高優先度のタスクは、割り込みハンドラの次に優先される。