6. Linux カーネルに関する知識 I

1. 科目の概要

OSS による OS の代表的な存在である Linux オペレーティングシステムを題材として、その内部構造の基本と、動作原理の基本的な概念を解説する。

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
I-6-1. 代表的なオペレーティングシステム	OSの基本的な役割を説明し、OSがコンピュータに搭載されるようになった経緯と現在に至る発展の歴史を解説する。さらに現在の代表的なOSを紹介し、それぞれのOSが持つ特徴や各OSの分類についても触れる。	1
I-6-2. OSの基本構造とカーネルの役割	Linux OSの基本的な構造を説明する。また中心的な役割を担うカーネルの基本的な機能として、プロセス管理、メモリ管理、ファイル管理、ネットワーク、I/Oなどの概要を紹介する。	2,5
I-6-3. マルチタスクOSにおけるプロセスの概念	「プロセス」の概念と、プロセスを切り替えて時分割でタスクを遂行するマルチタスクの概念について解説する。またOSが各プロセスを管理する手法についても言及する。	5
I-6-4. プロセスの切り替えとスケジューリングアルゴリズム	コンテキストスイッチの概念について触れ、プロセスが切り替わるメカニズムの概要を説明する。スケジューリングアルゴリズムの実際を紹介し、さらにその基礎となる理論である待ち行列議論とマルコフ連鎖について解説する。	2,5
I-6-5. 割り込みと時分割処理	CPUとその他デバイスの関係について整理し、「割り込み」の概念を説明する。割り込みの種類や割り込みの管理など割り込みに関連するいくつかの話題について触れ、さらに、カーネルにおける割り込み処理の概要を解説する。また時分割処理に関わる基本的な事項を説明する。	3
I-6-6. システムコール	一般のアプリケーションからOSの機能を利用する際に使用するシステムコールを紹介する。システムコールの位置づけを説明し、システムコールが呼ばれたときのOS内部での振る舞いについて概説する。	3,4
I-6-7. プログラムとOSの動作モード	プログラムが動作するプロセス空間という概念を説明し、その目的や特徴、それぞれの空間でできることとできないことを示す。さらにシステムコールによる動作モードの遷移について解説する。	4,5
I-6-8. 同期と排他制御	プロセスの同期や排他が必要になる状況(デッドロック等)やカーネルによる管理について述べ、排他制御を実現するための実装技術やその背景となる基礎理論について触れる。またカーネルの内部で排他制御が実装されている具体的な例を示して説明する。	4,5
I-6-9. プロセスのライフサイクル	プロセスが生成されてから終了して消滅するまでのライフサイクルについて解説する。プロセスの内部データ構造、状態遷移、プロセスグループ、プロセスを生成する方法など、プロセスのライフサイクルに関連する一連のトピックを紹介する。	2,5
I-6-10. プロセス間通信	プロセスの間でデータやメッセージを通信をするための手段として提供されている共有メモリ、セマフォ、メッセージなどを紹介する。	4,5

【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

「6. Linux カーネルに関する知識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
6. Linuxのカーネルに関する知識	<Linuxカーネル概論>	<スケジューリング>	<割り込みと遅延>	<システムコール>	<プロセス管理>	<メモリ管理(1)>	<メモリ管理(2)>	<メモリ管理(3)>	<ファイル管理(1):仮想ファイルシステム>	<ファイル管理(2):ファイルの操作>	<ファイル管理(3):特殊ファイル>	<ネットワーク(1):ソケットインタフェース>	<ネットワーク(2):IPとUDP>	<ネットワーク(3):UDPとTCP>	<ネットワーク(4):TCPフロー制御と輻輳制御>

[シラバス：http://www.ipa.go.jp/software/open/ossc/download/Model_Curriculum_05_06.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
組織関連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WAN	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 [6-1-1,2]	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 [6-1-2]	CE-OPS1. 並行性 [6-1-1,2,4]	CE-OPS2. スケジューリングとディスパッチ [6-1-3,4,5,6]	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 [6-1-2,5]	CE-OPS7. OSの概要	CE-OPS8. 設計の原則 [6-1-1,3]	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	13	CE-CAO コンピュータのアーキテクチャと構成	CE-CAO0. 歴史と概要	CE-CAO1. コンピュータアーキテクチャの基礎	CE-CAO2. メモリシステムの構成とアーキテクチャ	CE-CAO3. インタフェースと通信	CE-CAO4. デバイスサブシステム	CE-CAO5. CPUアーキテクチャ	CE-CAO6. 性能・コスト評価	CE-CAO7. 分散・並列処理	CE-CAO8. コンピュータによる計算	CE-CAO9. 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-ESY13. リアルタイムシステム設計	CE-ESY14. 組込みマイクロコントローラ	CE-ESY15. 組込みプログラム	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	CE-ESY12. 実装

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として、Linux カーネルの基本構造と起動プロセスとがある。他の多くの項目は一般的なオペレーティングシステムの機能について Linux を通して習得するものである。

科目名	第1回	第2回	第3回	第4回	第5回
6.Linux カーネルに関する知識Ⅰ	(1)OS の歴史とLinux (2)OS の概念 (3)Linux OS の基本構造 (4)カーネルの基本機能 (5)カーネルの起動プロセス (6)参考	(1)マルチタスキング (2)プロセス (3)コンテキストスイッチング(プロセスディスパッチ) (4)プロセススケジューラ (5)マルチプロセッサへの対 (6)スケジューリングのアルゴリズム (7)基礎理論	(1)割り込み (2)タイマー	(1)システムコール (2)同期と排他	(1)プロセス管理 (2)シグナル (3)プロセス間通信

(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
システム分野	6　Linux カーネルに関する知識 I	基本
習得ポイント	I-6-1. 代表的なオペレーティングシステム	
対応する コースウェア	第1回 (Linux カーネル概論)	

I-6-1. 代表的なオペレーティングシステム

OS の基本的な役割を説明し、OS がコンピュータに搭載されるようになった経緯と現在に至る発展の歴史を解説する。さらに現在の代表的な OS を紹介し、それぞれの OS が持つ特徴や各 OS の分類についても触れる。

【学習の要点】

＊ OS(Operating System)とは基本ソフトウェアの一つであり、基本的な役割として、プログラム間の資源管理や、ハードウェアへのインタフェースの提供などがある。

＊ 1950年代の初期のコンピュータは、オペレーティングシステムを持たなかったが、1960年代にマルチプログラミング・タイムシェアリングシステムなどが登場して、これらの概念を搭載するオペレーティングシステムが登場した。

＊ 1960年代後半にハードディスクドライブが登場したことで、著しい進化を遂げてゆく。

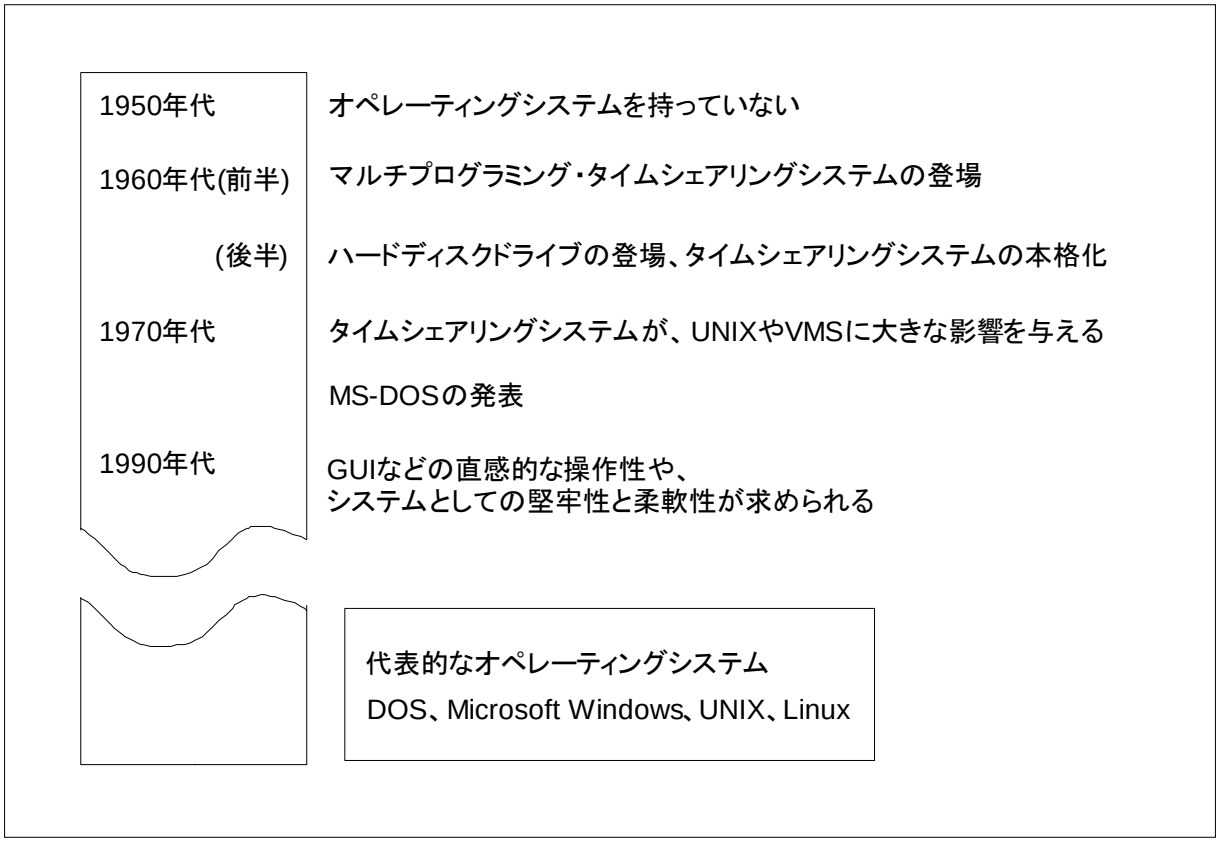

図 I-6-1.　オペレーティングシステム発展の歴史

【解説】

1) オペレーティングシステムの基本的な役割

オペレーティングシステムとは、基本ソフトウェアの1つである。基本的な役割は以下の通り。

＊ 資源管理
コンピュータ上で複数個のアプリケーションプログラムが同時に起動された場合に、互いのプログラムが相手の資源に干渉しないように管理する。

＊ ハードウェアの抽象化
アプリケーションプログラムに対して、様々なハードウェアへの統一的で単純化された利用インタフェースを提供する。

＊ コンピュータの利用効率の向上
複数個のプログラムを同時に起動させる場合に、資源の割り当てや処理の順番、および処理の割り当て時間を工夫することで、全体のデータ処理容量を向上させる。

2) オペレーティングシステム発展の歴史

＊ 第1世代
1950 年代。初期のコンピュータはオペレーティングシステムを持っていなかったが、システム管理用ソフトウェアツールなどは早々に開発が行われ、次第にその利用範囲を広げていった。

＊ 第2世代
1960 年代前半。マルチプログラミング・タイムシェアリングシステム・ジョブ管理・仮想記憶の概念が登場し、これらの概念を搭載するオペレーティングシステムが登場してきた。

＊ 第3世代
1960 年代後半。ハードディスクドライブの登場とタイムシェアリングシステムの本格的な実用化などにより、著しい進化を遂げてゆく。

＊ 第4世代
1970 年代。Multics のタイムシェアリングシステムが、UNIX や VMS などに大きな影響を与えてゆく。パソコン向けの OS として CP/M が登場し、これのクローンとして MS-DOS が発表される。

＊ 第5世代
1990 年代。GUI とともにメインフレーム用途の OS の、堅牢性と柔軟性が求められる。この頃には、パーソナルコンピュータがこのような要請に応えられるだけの性能を持っていた。

3) 代表的なオペレーティングシステム

＊ DOS（MS-DOS、PC-DOS、CP/M、DR-DOS など）
ディスク装置を前提としたコンピュータのオペレーティングシステムである。

＊ Microsoft Windows
米国マイクロソフトが開発及びライセンス販売を行うコンピュータのオペレーティング環境。

＊ UNIX（AIX、BSD、System Ⅴ、Solaris）
マルチタスク、マルチユーザ機能を有する OS の1つ。また、これから派生した一連の OS の総称。

＊ Linux
UNIX によく似たコンピュータ用オペレーティングシステム。

スキル区分	OSS モデルカリキュラムの科目	レベル
システム分野	6　Linux カーネルに関する知識 I	基本
習得ポイント	I-6-2. OS の基本構造とカーネルの役割	
対応する コースウェア	第2回（スケジューリング） 第5回（プロセス管理）	

I-6-2. OS の基本構造とカーネルの役割

Linux OS の基本的な構造を説明する。また中心的な役割を担うカーネルの基本的な機能として、プロセス管理、メモリ管理、ファイル管理、ネットワーク、I/O などの概要を紹介する。

【学習の要点】

＊ LinuxOS の基本的な構造としては、中心にカーネルと呼ばれるプログラムが存在し、そのカーネルを取り囲むようにしてシェル・カーネルモジュール・アプリケーションプログラムが存在する。

＊ カーネルは、LinuxOS の中核となるプログラムで、プログラムをメモリに読み込んだり、プログラム実行に必要な領域を確保したりする。その他にも、ファイルや周辺機器の管理や、ネットワークと I/O の管理などを行う。

＊ シェルとは、ユーザから与えられた指示をカーネルが解釈できる形式に変換するプログラムである。

＊ カーネルモジュールとは、必要に応じて追加されるプログラム部品である。

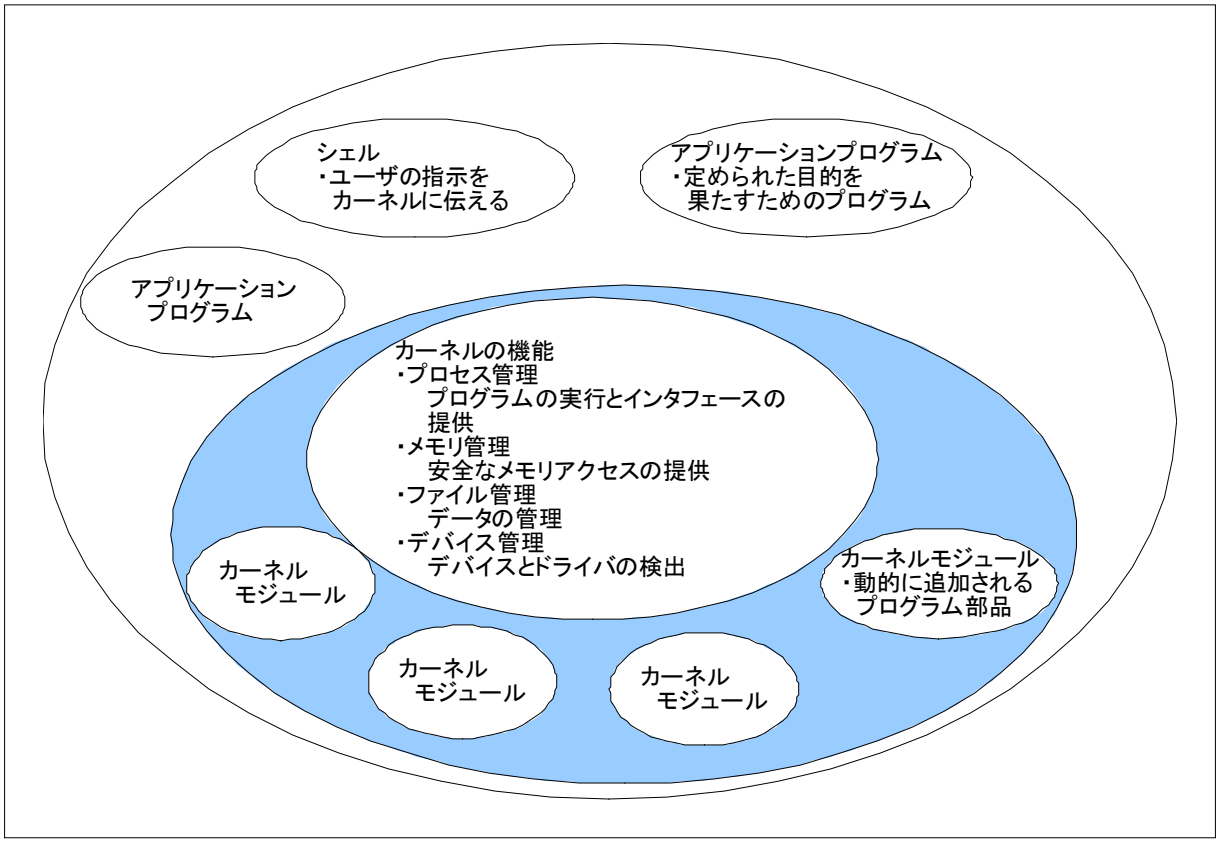

図 I-6-2.　OS の基本構造

【解説】

1) Linux オペレーティングシステムの基本的な構造

LinuxOS には、その中心にカーネルと呼ばれる核となるプログラムが存在している。そのカーネルを中心に、シェルやカーネルモジュール、アプリケーションプログラムなどによって構成されている。

＊ シェルとは
ユーザからの指示・命令を受けて、それらをカーネルが解釈できる形式に変換して伝えるソフトウェアのことである。

＊ カーネルモジュールとは
必要に応じて追加されるプログラム部品。デバイスドライバなどはこれに該当する。

＊ アプリケーションプログラムとは
コンピュータの OS の上で実行され、ユーザの目的を果たすための機能を提供するプログラムのことを示す。ただし、ミドルウェアなどは含まれない。

2) カーネルの基本的な機能

カーネルとは、OS の核となるプログラムである。基本的な機能は、以下の通りである。

＊ プロセス管理
プロセス管理とは、アプリケーションプログラムの実行を許可し、ハードウェアへのアクセスのためのインタフェースをプログラムに提供することである。プログラムを実行するため、カーネルはプログラムのコードを含むファイルをメモリに読みこみ、プログラム実行に必要な専用領域（スタックと呼ぶ）を準備し、そのプログラムの所定の位置へ制御を渡すことで実行を開始する。

＊ メモリ管理
カーネルはアプリケーションプログラムからの要求のたびに、アクセス可能なメモリ空間を提供する。これは仮想アドレッシングと呼ばれる方式であり、この方式ではカーネルは物理アドレスを別のアドレス（仮想アドレス）に変換する。

＊ ファイル管理
ファイル管理（ファイルシステム）は、コンピュータの資源を管理するための、OS が持つ機能の1つである。ファイルとは、ハードディスクなどの補助記憶装置に格納されたデータを示している。

＊ デバイス管理
デバイス管理では、各種のデバイスにアプリケーションシステムからアクセスするための統一的なインタフェースや、 デバイスドライバの管理を行なう。オペレーティングシステムの起動時、カーネルはバス上の周辺機器を検索し、必要なドライバを探す。

＊ I/O 管理
外部からのコンピュータへのデータ入力、コンピュータから外部へのデータ出力の管理も行っている。

スキル区分	OSS モデルカリキュラムの科目	レベル
システム分野	6　Linux カーネルに関する知識 I	基本
習得ポイント	I-6-3. マルチタスク OS におけるプロセスの概念	
対応する コースウェア	第5回（プロセス管理）	

I-6-3. マルチタスク OS におけるプロセスの概念

「プロセス」の概念と、プロセスを切り替えて時分割でタスクを遂行するマルチタスクの概念について解説する。また OS が各プロセスを管理する手法についても言及する。

【学習の要点】

* プロセスとは、オペレーティングシステム上における処理の単位の1つであり、実行命令コードやプロセス固有データ・リソース記述子などで構成されている。
* オペレーティングシステムが、複数のプロセスを切り替えながら実行することをマルチタスクという。この場合、ユーザ側から見ると複数のプログラムが同時に実行されているように見える。
* プロセス管理は、カーネルの重要な機能の1つである。

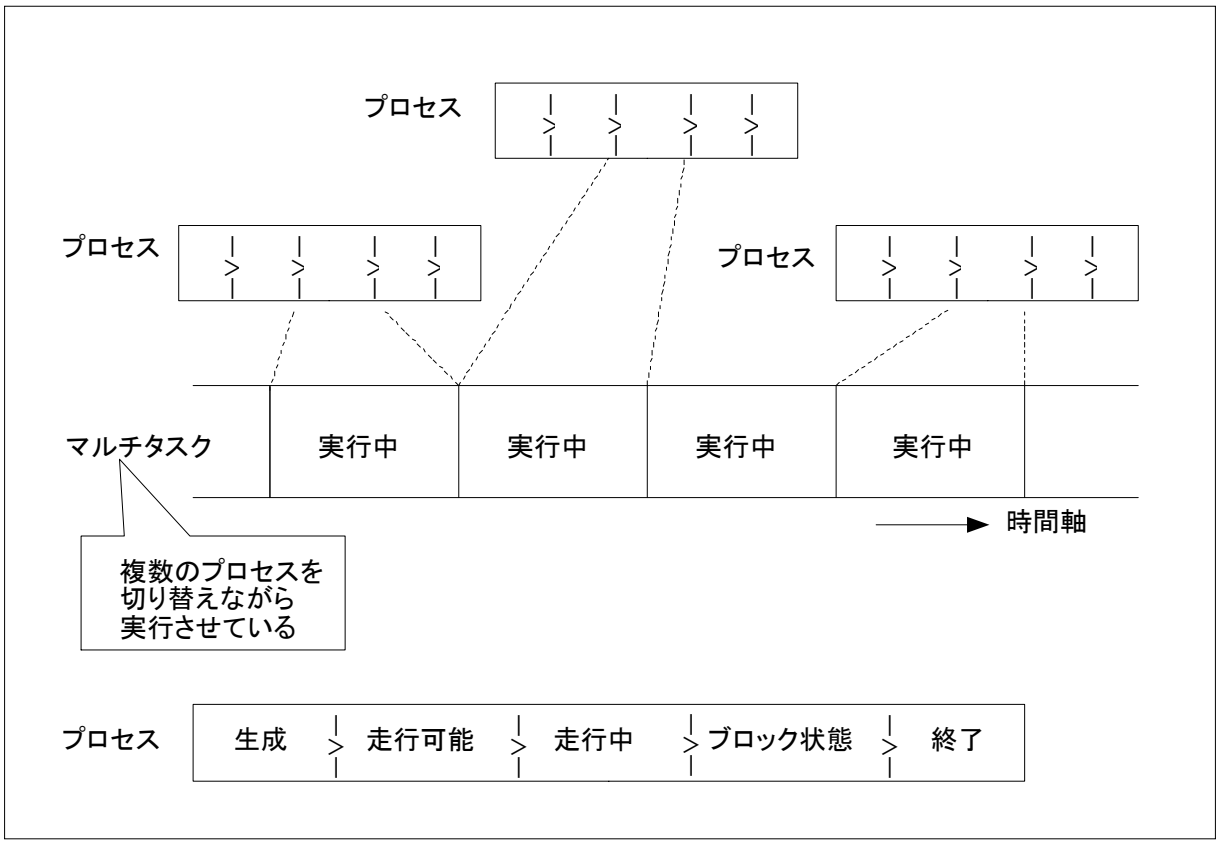

図 I-6-3.　マルチタスクの概念

【解説】

1) プロセスとは

プロセスとは、オペレーティングシステム上における処理の単位の1つである。マルチタスクオペレーティングシステムでは、多くのプロセスを並列的に実行させることが可能である。また、「ジョブ」と表現されることもあるが、「ジョブ」がユーザ側から見た処理の単位であり、「プロセス」はコンピュータ側から見た処理の単位を表している。プロセスは以下に示す資源で構成されている。

* 実行命令コード
* プロセス固有データ
* リソースの記述子
* セキュリティ属性
* プロセッサ状態

2) マルチタスクの概念

オペレーティングシステムが、複数のプロセスを切り替えながら実行することをマルチタスクという。プロセスが順次切り替わりながら実行されていく仕組みだが、ユーザ側から見ると複数のプログラムが同時に実行されているように見える。また、プロセスが入出力待ち状態となっても他のプロセスが実行されるため、全体として処理速度が向上する。

* プリエンプティブ・マルチタスク
  プリエンプティブ・マルチタスクとは、プロセスの切り替えの際にハードウェアによるタイマ割り込みが用いられる方法である。この割り込みによって制御を OS に戻す。これを強制的に繰り返している。
* ノンプリエンプティブ・マルチタスク
  各プロセス自身が、短い時間間隔で OS に制御を戻すことによりマルチタスクが実現されているものを、ノンプリエンプティブ・マルチタスクと呼ぶ。

3) プロセスの管理方法

プロセス管理とは、カーネルの重要な機能の1つである。(ユーザ)プロセスの様々な状態はカーネルプロセスにより管理されている。プロセスが最初に生成されると「生成」もしくは「新規」という状態となり、そのあとで、実行待ち(待機)、実行中といった状態に遷移しながら、最終的には「終了状態」へと遷移してゆく。このような状態遷移を、複数のプロセス間の状況に基づき、スケジューリングし、コントロールしているのがカーネルプロセスである。

スキル区分	OSS モデルカリキュラムの科目	レベル
システム分野	6　Linux カーネルに関する知識 I	基本
習得ポイント	I-6-4. プロセスの切り替えとスケジューリングアルゴリズム	
対応する コースウェア	第2回（スケジューリング） 第5回（プロセス管理）	

I-6-4. プロセスの切り替えとスケジューリングアルゴリズム

コンテキストスイッチの概念に触れ、プロセスが切り替わるメカニズムの概要を説明する。スケジューリングアルゴリズムの実際を紹介し、さらにその基礎となる理論でもある待ち行列議論とマルコス連鎖について解説する。

【学習の要点】

＊ マルチタスク・オペレーティングシステムにおいて、プロセスの切り替えを行う際には、コンテキストスイッチが発生する。これは、CPU の状態を保存したり復元したりする過程のことである。

＊ スケジューラは、プロセスの切り替え先を決定するプログラムで、各プロセスはタイムスライスと呼ばれる短い時間ずつ実行されたあと、切り替えられる。

＊ スケジューラは、CPU の負荷を公平かつ平等に分散させるために、スケジューリングアルゴリズムに基づいて、プロセスの制御を行う。

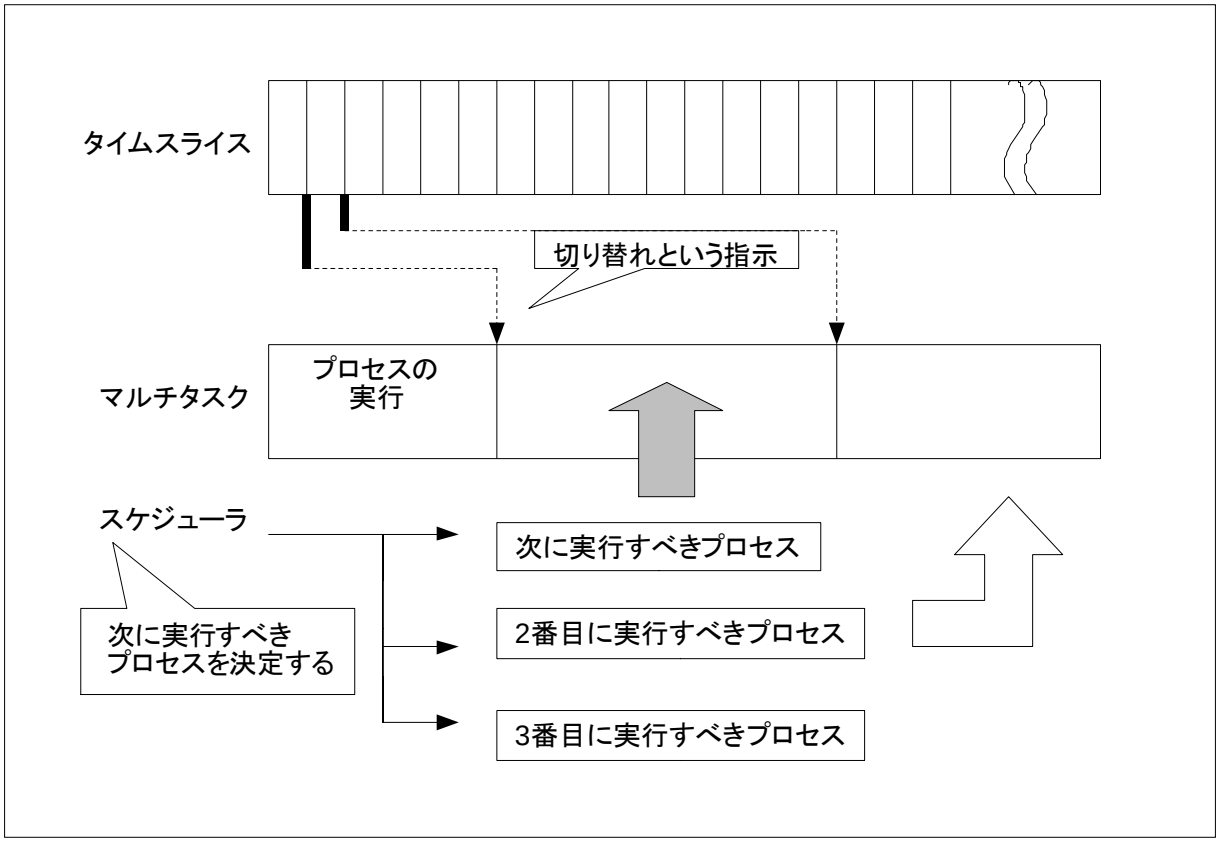

図 I-6-4.　プロセスの切り替え

【解説】

1) プロセスが切り替わるメカニズム

マルチタスク・オペレーティングシステムにおいては、実行中のプロセスであっても途中で CPU の使用を止め、他のプロセスに切り替えてそのプロセスに CPU の使用が可能となるようにしている。このプロセスの切り替えの際にコンテキストスイッチが発生する。

＊ コンテキストスイッチとは
プロセスの切り替えの際に、それまでの CPU の状態を保存したり、もしくは復元したりする処理のことである。

＊ タイムスライスとは
プリエンプティブ・マルチタスクでは、スケジューラが各プロセスを定められた時間だけ実行させて、他のプロセスに切り替えている。この時間のことをタイムスライスと呼ぶ。

＊ スケジューラ
実行プロセスの切り替え先を決定するプログラムがスケジューラである。スケジューラは、優先度つきキューで優先度を割り当てられたプロセスの制御を行う。

＊ 割り込み
プロセスがタイムスライスの時間内で終了しなかった場合には、タイマ割り込みが発生して他のプロセスへの切り替えが行われる。

2) スケジューリングアルゴリズムとは

スケジューラの目的は、CPU の負荷を分散し、全体として効率的な稼動を確保することにある。スケジューラは、スケジューリングアルゴリズムに基づいてプロセスの制御を行う。一般的に知られているスケジューリングアルゴリズムには、以下のようなものがある。

＊ FIFO （ First In, First Out ）
「先入れ先出し」方式のこと。最初に発生したリクエストをキューに並べ、そのリクエストの処理が終了してから、次に発生したリクエストの処理を開始する。

＊ LIFO （ Last In, First Out ）
あとから発生したリクエストを、先に処理する方法。公平なスケジューリングは実現されない。

＊ ラウンドロビンスケジューリング
キューに並んでいる処理待ち状態のプロセスに対して、特に優先度などの設定は行わず、順番に同じタイムスライスを割り当てていく。

＊ 多段フィードバックキュー
短いプロセスや対話型のプロセスを優先し、プロセスの性質を迅速に把握しスケジューリングを行う。UNIX の基本的なアルゴリズムである。

3) スケジューリングアルゴリズムの基礎となる理論

スケジューリングを行う際、各プロセスはキューと呼ばれる「コンピュータの基本的なデータ構造」の中に並べられる。キューは待ち行列とも呼ばれる。

スキル区分	OSS モデルカリキュラムの科目	レベル
システム分野	6　Linux カーネルに関する知識 I	基本
習得ポイント	I-6-5. 割り込みと時分割処理	
対応する コースウェア	第3回（割り込みと遅延）	

I-6-5. 割り込みと時分割処理

CPU とその他デバイスの関係について整理し、「割り込み」の概念を説明する。割り込みの種類や割り込みの管理など割り込みに関連するいくつかの話題について触れ、さらに、カーネルにおける割り込み処理の概要を解説する。また時分割処理に関わる基本的な事項を説明する。

【学習の要点】

* 割り込みが発生すると、CPU は現在実行中の処理を中断して、割り込みに対応する処理を実施する。
* 周辺機器に何らかの障害が発生した場合に、割り込みを発生させて障害を伝えることにより、回復処理などの実行を速めることが可能となる。
* タイムシェアリングシステムにおいては、タイムスライスの時間内に CPU の使用が止まらなかった場合に、タイマ割り込みが発生する。

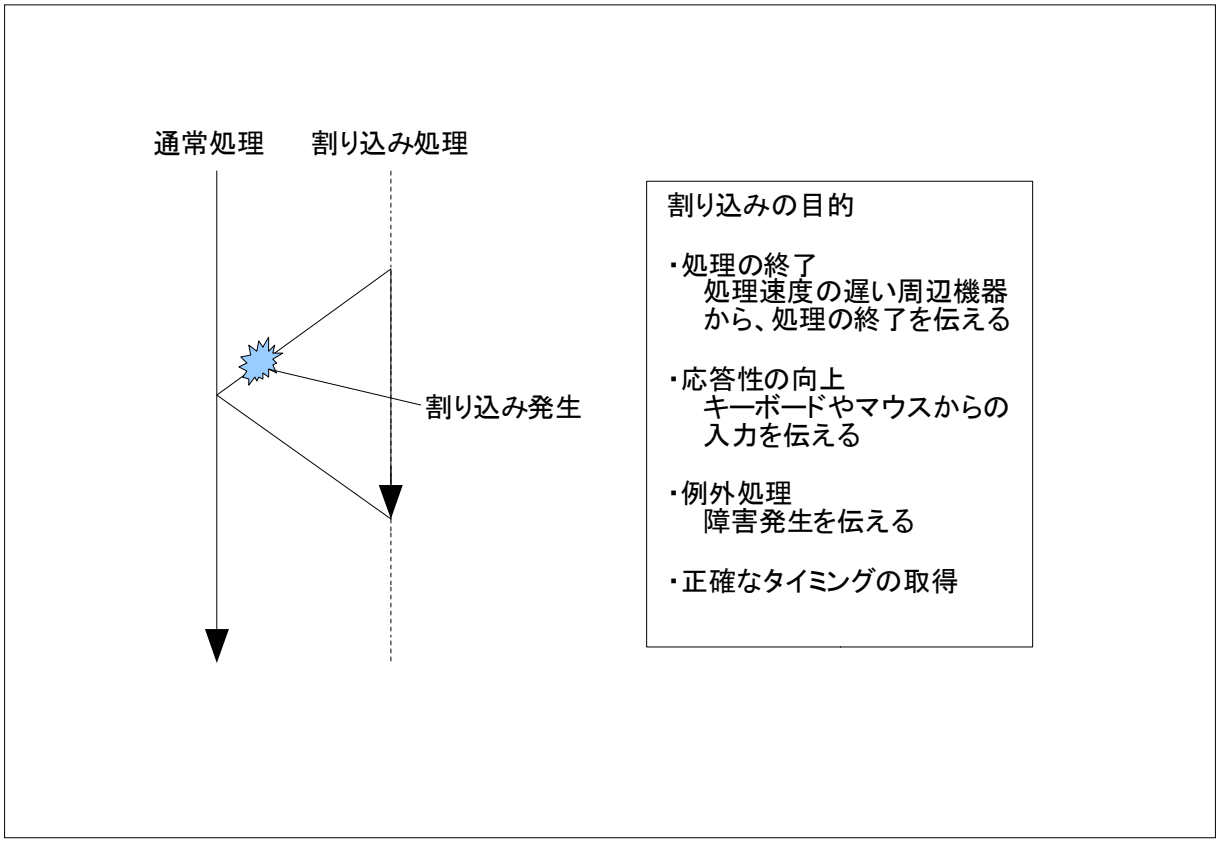

図 I-6-5.　割り込みの目的

【解説】

1) CPU が周辺機器からの要求を受け取る方法

コンピュータが周辺機器などから受け取る要求のことを割り込みと呼ぶ。コンピュータは割り込みが発生すると現在実行中の処理を中断して、割り込みに対応するための処理を行う。割り込みの目的を以下に記す。

* 処理速度の遅い周辺機器に計算リソースを独占されることのないように、割り込みによって CPU は他の処理を効率よく行うことが可能となる。ただし、キーボードやマウスなどの直接ユーザが操作するデバイスでは、割り込みを使ってユーザの入力1つ1つを確実に処理させている。
* 周辺機器に何らかの障害が発生した場合に、割り込みを発生させて障害を伝えることにより、回復処理など必要な処理の実行を早めることが可能となる。
* 一定のタイミングで処理することが必須条件であるような機器の制御を行う場合に、所定のスケジュールでのタイマ割り込みを発生させ、CPU 側に処理のタイミングを指示することを可能とする。

2) 割り込みの種類

CPU の割り込みは、大きくわけて「ハードウェア割り込み」と「ソフトウェア割り込み」に分類できる。

* ハードウェア割り込み
  CPU の外部から、CPU の割り込み要求端子に対して電気的な信号変化を与えることで発生する。
* ソフトウェア割り込み
  実行中のソフトウェアによる割り込み。ゼロ除算やオーバーフローなどによって発生する。

3) 割り込みの管理

ハードウェア割り込みにせよソフトウェア割り込みにせよ、CPU に割り込みが生じると、現在実行しているプロセスを中断させ割り込みハンドラもしくは割り込みサービスルーチンと呼ばれる処理が実行される。また、これら割込処理の終了後には、元のプロセスが中断した箇所から再開されるように、元のプロセスに関する情報(アドレスや作業状態)を、保存しておく。この過程をコンテキストスイッチという。

* タイマ割り込み
  タイムシェアリングシステムにおいては、プロセスがタイムスライスの時間内に CPU の使用を止めなかった場合には、タイマ割り込みが発生し他のプロセスの実行が開始される。この場合にも、同様の割り込み管理が行われている。

スキル区分	OSS モデルカリキュラムの科目	レベル
システム分野	6　Linux カーネルに関する知識 I	基本
習得ポイント	I-6-6. システムコール	
対応する コースウェア	第3回（割り込みと遅延） 第4回（システムコール）	

I-6-6. システムコール

一般のアプリケーションから OS の機能を利用する際に使用するシステムコールを紹介する。システムコールの位置づけを説明し、システムコールが呼ばれたときの OS 内部での振る舞いについて概説する。

【学習の要点】

- ＊ システムコールとは、アクセスできない保護領域にアクセスすることや、保護されたレジスタを操作するなど、カーネルの持つ特殊な機能を利用する場合に使用する仕組みのことである。
- ＊ Linux では約300種類のシステムコールが用意されており、CPU の動作モードなどを変更することも可能である。
- ＊ システムコールは、そのほとんどがソフトウェア割り込みによって実行されているため、その処理方法は一般の割り込み処理と同様である。

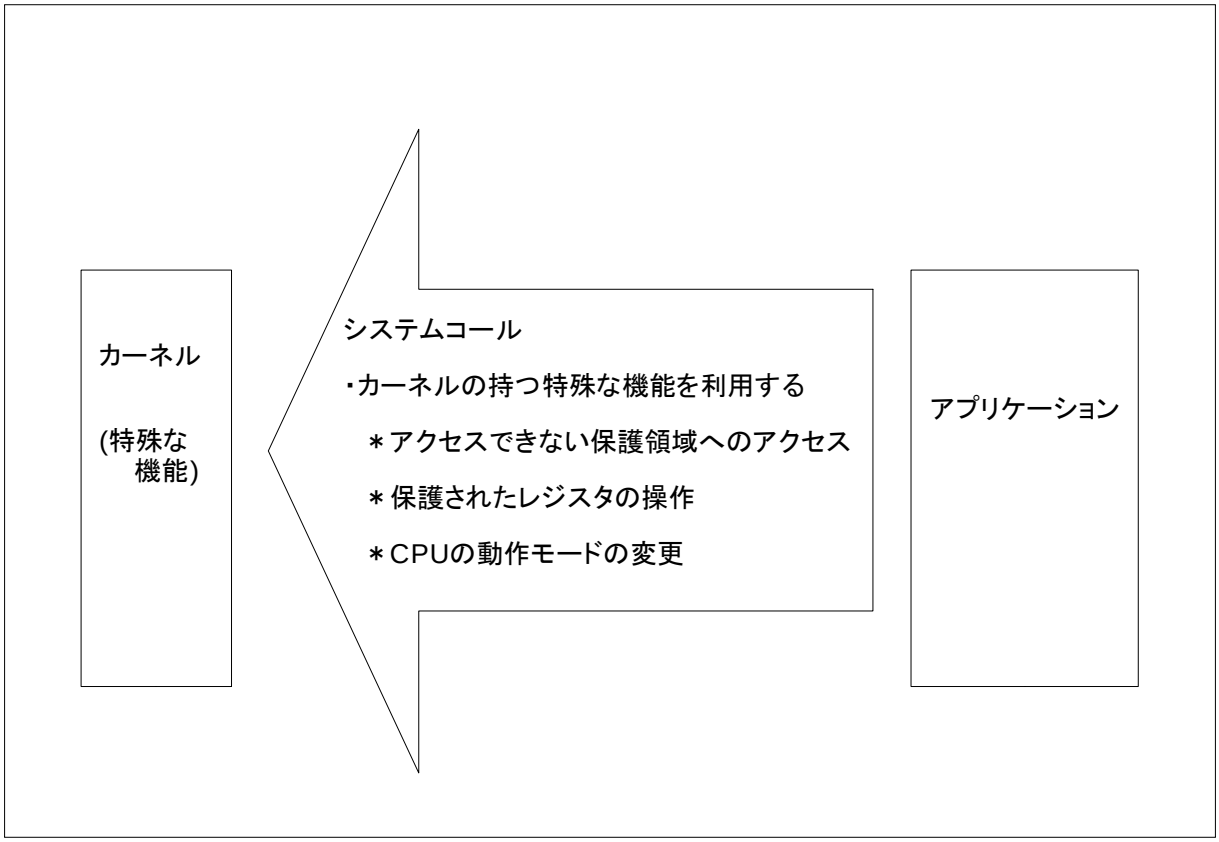

図 I-6-6.　システムコール

【解説】

1) システムコールとは

カーネルが提供する機能を一般のアプリケーションから利用する際に呼び出すサービスのことをシステムコールという。

＊　システムコールでできること

- 通常のアプリケーションからはアクセス不可能な保護対象のメモリ領域へのアクセス
- 通常は保護されているレジスタの操作
- CPU の動作モードの変更(一般のアプリケーションは勝手に変更できない)

＊ システムコールの数

OS によって異なるが、Linux の場合、300 個程度の種類が用意されている。FreeBSD の場合は、およそ 330 個のシステムコールを利用可能である。

＊ システムコールの例

open, read, write, close, wait, fork, execve, kill といったシステムコールが POSIX で定義されている。なお POSIX とは、様々な UNIX の実装やその他の OS 間で共通な API を定め、アプリケーションの移植性を向上させることを目的として策定されたアプリケーションプログラムインタフェースの規格である。IEEE によって定められており、システムコールの定義も POSIX で用意されている。

2) システムコールの処理方法

システムコールは、ソフトウェア割り込みで実現されることが多い。したがってシステムコールが発効された際の動作は、ソフトウェア割り込みに対する処理と同様となる。

＊ システムコールが呼び出されたときの動作

- システムコールを呼び出したプロセスはその時点で中断
- その時点でのコンテキストを保存
  （システムコールからの復帰時にプロセスを正しく再開できるように）
- 呼び出された番号からシステムコールを選択、引数を設定
  （なおこの際に呼び出したプロセスがそのシステムコールを実行する権利を持っているかどうかを調べ、問題がなければそのまま続行する）
- 選択されたシステムコールを実行
- システムコールの処理終了後、保存したコンテキストに従い呼び出し元プロセスに復帰

＊ プロセス再開時の実行順

システムコール終了後、呼び出したプロセスが直ちに再開されない場合がある。システムコールを呼び出したプロセスは、システムコールの処理終了後に復帰できるよう「実行可能」キューに並べられる。実行可能キューには複数のプロセスが入れられる可能性があり、このような場合、システムコールの処理が終了してもそのシステムコールを呼び出したプロセスが、終了後すぐに再開されるわけではない。

スキル区分	OSS モデルカリキュラムの科目	レベル
システム分野	6　Linux カーネルに関する知識 I	基本
習得ポイント	I-6-7. プログラムと OS の動作モード	
対応する コースウェア	第4回（システムコール） 第5回（プロセス管理）	

I-6-7. プログラムと OS の動作モード

プログラムが動作するプロセス空間という概念を説明し、その目的や特徴、それぞれの空間でできることとできないことを示す。さらにシステムコールによる動作モードの遷移について解説する。

【学習の要点】

* マルチタスク機能であれば、複数のプロセスがタイムスライスとして定められた時間で切り替わりながら、処理を行っている。そのため、処理が完全に終了していないプロセスに関しては、何度も再開されることになる。
* プロセス毎に割り当てられたアドレス空間のことを、プロセス空間とよぶ。他のプロセスからは一切参照されることがないため、データの破壊などは発生しない。
* Linux カーネルはスーパーバイザモードで動作し、一般のアプリケーションはユーザモードで動作している。アプリケーションプログラムによりシステムコールが呼び出された場合、CPU の動作モードはスーパーバイザモードへ遷移する。

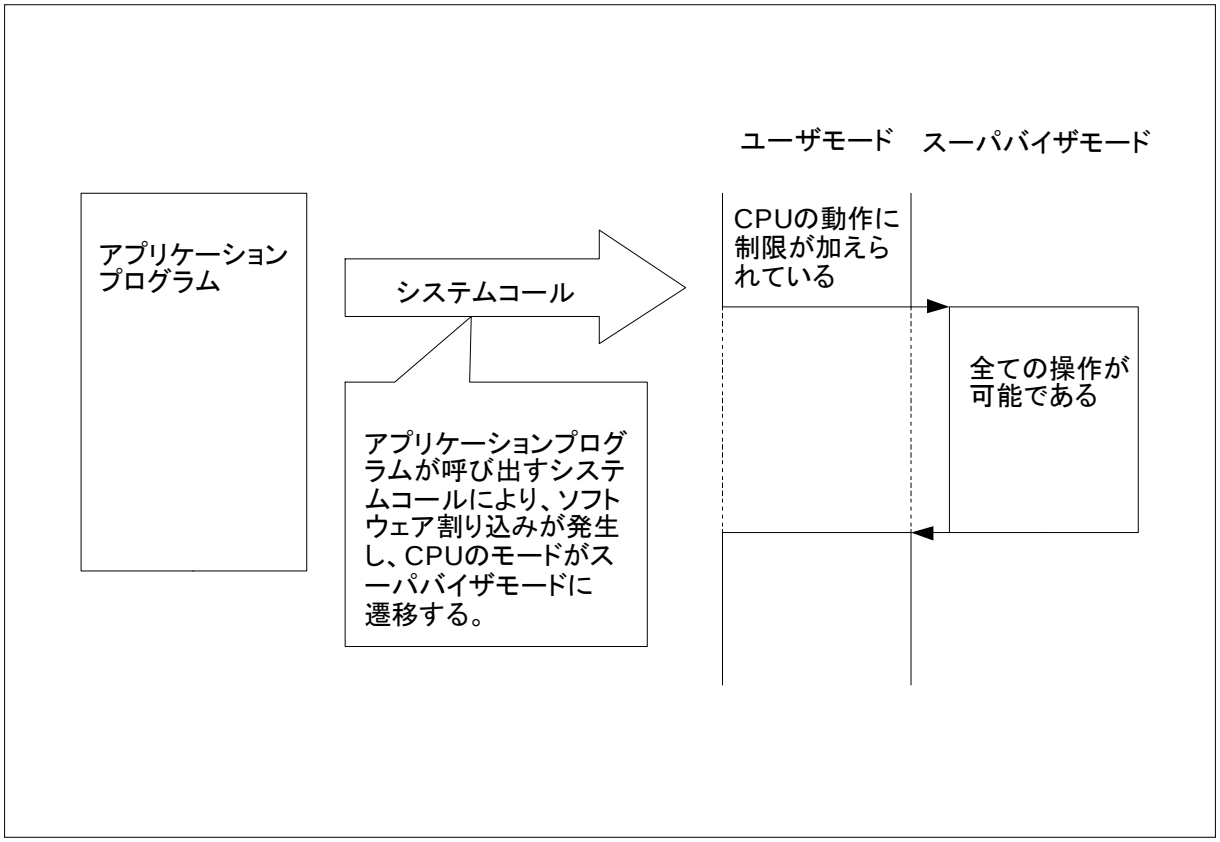

図 I-6-7.　OS の動作モードとシステムコールによる遷移

【解説】

1) プログラムの動作

Linux カーネルはマルチタスク機能を提供している。そのため、同時に複数のプロセスを実行することが可能である。

＊ プログラムとプロセスの関係
プログラムが実行されている状態とは、プロセスが実行されている状態である。ただし、1プログラムにつきプロセスは1つだけとは限らない。1つのプログラムを起動させることで、複数のプロセスが起動するものもある。

＊ 複数のプロセスを同時に起動させるということ
マルチタスク機能が提供されているため、同時に複数のプロセスの実行が可能ではあるが、実際にある一時点で動作しているプロセスの数は、システムに搭載されている CPU 数以上には決してならない。実際はマルチタスク機能が、複数のプロセスを細かく切り替えながら動作させ、いかにも同時動作しているような環境を作り出しているだけである。

＊ 複数のプロセスを切り替えながら動作させるということ
プリエンプティブ・マルチタスクであれば、タイムスライスとして定められた時間でプロセスの切り替えが行われる。ほかにも割り込みの発生などでもプロセスの切り替えが発生する。

＊ 切り替えられたプロセスを再開するには
処理が完全に終了していないプロセスは、マルチタスクの機能により何度でも再開されることになる。このため、正常にプロセスを再開させるために個々のプロセスは固有のコンテキストを保存している。コンテキストとは、そのプロセスが動作するためのプロセス空間、そのプロセスが動作するときのレジスタ値などである。

＊ プロセス空間とは
プロセス毎に割り当てられる独立した論理アドレス空間のことをプロセス空間と呼ぶ。個々のプロセスからアクセス可能なメモリ空間とは、このプロセス空間のことであり、他のプロセスからは一切参照することはできない。プロセス間ではデータの授受は行えないが、他のプロセスからデータを破壊されることもない。

2) 動作モードの遷移

Linux カーネルは CPU の持っているスーパーバイザモードで動作している。また、一般のアプリケーションプログラムはユーザモードで動作している。このように CPU はいくつかの特権モードを持ち、そのモード毎に用意された命令を実行している。また、一般のアプリケーションプログラムにより呼び出されるシステムコールは、ソフトウェア割り込みを発生させて CPU の動作モードをスーパーバイザモードへ変更することが可能である。

＊ スーパーバイザモード
全ての命令の実行が可能で、かつ全てのメモリ空間にアクセスすることが可能なモード。特権モードと呼ばれることもある。

＊ ユーザモード
スーパーバイザモードの付帯機能と考えられるモード。一部の命令については実行できなかったり、メモリ空間の一部にはアクセスできなかったりする。

スキル区分	OSS モデルカリキュラムの科目	レベル
システム分野	6　Linux カーネルに関する知識 I	基本
習得ポイント	I-6-8. 同期と排他制御	
対応する コースウェア	第4回（システムコール） 第5回（プロセス管理）	

I-6-8. 同期と排他制御

プロセスの同期や排他が必要になる状況（デッドロック等）やカーネルによる管理について述べ、排他制御を実現するための実装技術やその背景となる基礎論理について触れる。またカーネルの内部で排他制御が実装されている具体的な例を示して説明する。

【学習の要点】

＊ マルチタスクは、排他制御機能を利用して資源の保護を行っているが、排他制御には欠陥があり、デッドロックなどが発生してしまう可能性がある。

＊ システム構築の際には、オブジェクト指向プログラミングなどを利用して、デッドロックが発生しないように注意すべきである。

＊ 排他制御を実現するための実装技術には、様々なものがある。また、カーネル内部でも排他制御は様々な箇所で使用されている。

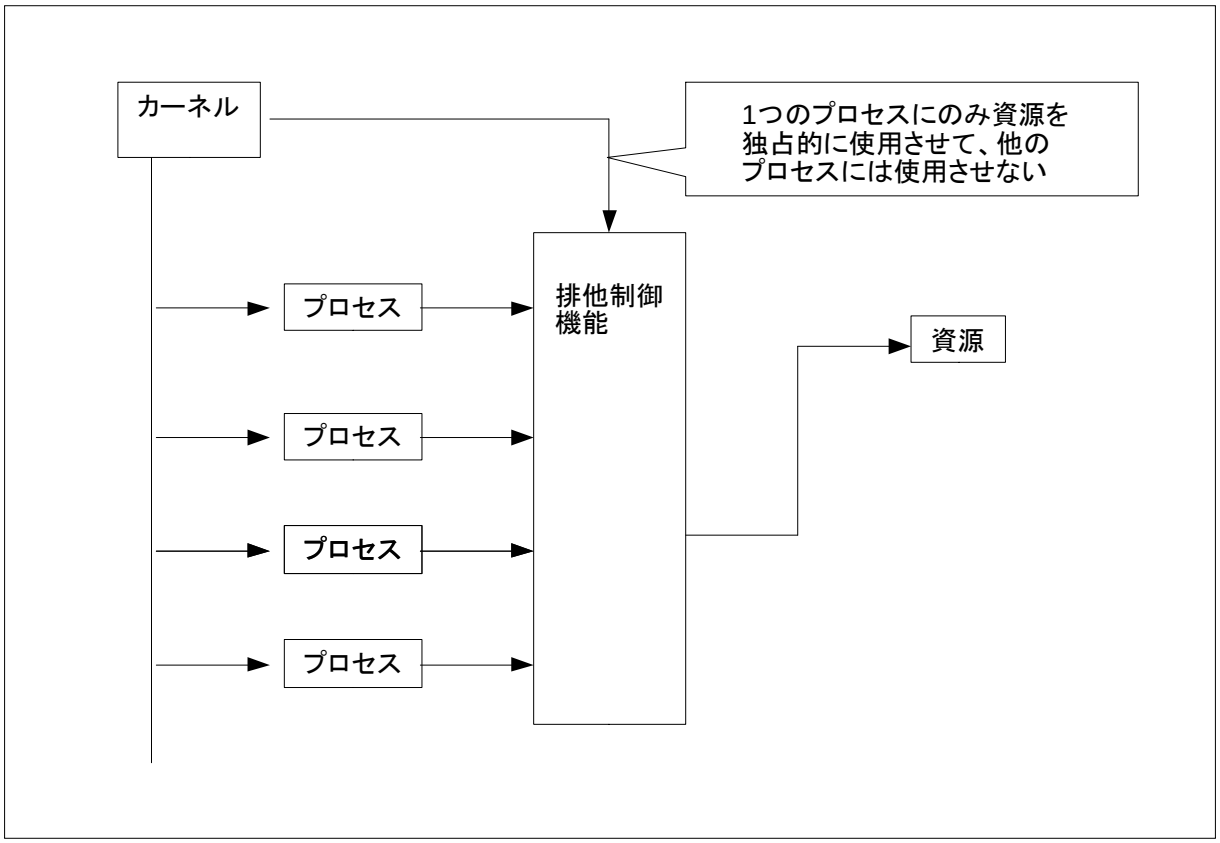

図 I-6-8.　排他制御

【解説】

1) デッドロックとは

デッドロックとは、2つ以上のプロセスが実行されている状態において、お互いに相手の処理が終了することを待つという状態となってしまい、外部から見るとプロセスが停止しているように見えることをいう。

* デッドロックは、排他制御の欠陥である。
  マルチタスク機能においては、コンピュータ上の一つの資源に対して、同時に複数の要求が発行された場合に、全てのプロセスに対して排他制御を要求する。その結果としてデッドロックが発生してしまう。
* プロセス間の同期。
  マルチタスク機能は、特定のプロセスが何らかのイベントを起こすまで、その他のプロセスに処理の続行を待たせることで、プロセス間の同期をとる場合がある。
* 排他制御とは。
  複数のプロセスからの同時アクセスにおいて、データの整合性を保つために、1つのプロセスにのみ独占的に資源を利用をさせ、他のプロセスが利用できないようにすることを排他制御と呼ぶ。
* デッドロックを回避するため方法として以下のようなものがある。
  - オブジェクト指向プログラミングを用いてシステムを構築する。
  - Lock-free と Wait-free アルゴリズムを用いる。

2) 排他制御を実現するための実装技術

排他制御を実現する場合には、以下に示す処理を実施する必要がある。ただし、この操作の途中で他に制御が渡ってしまうと排他制御は実現できない。

* 資源が他のプロセスによって占有されていないことを確認する。
* 資源を占有する。

排他制御を実現する方法としては以下のものがある。

* ミューテックス
* セマフォ
* 同期
* ロック
* スピンロック
* タスク切り替え禁止
* 割り込み禁止

3) カーネル内部での排他制御

カーネル内部でも排他制御は様々な箇所で使用されている。メモリ管理、デバイス管理、およびマルチプロセッサ管理などである。

スキル区分	OSS モデルカリキュラムの科目	レベル
システム分野	6　Linux カーネルに関する知識 I	基本
習得ポイント	I-6-9. プロセスのライフサイクル	
対応する コースウェア	第2回（スケジューリング） 第5回（プロセス管理）	

I-6-9. プロセスのライフサイクル

プロセスが生成されてから終了して消滅するまでのライフサイクルについて解説する。プロセスの内部データ構造、状態遷移、プロセスグループ、プロセスを生成する方法など、プロセスのライフサイクルに関連する一連のトピックを紹介する。

【学習の要点】

* Linux 上の全てのプロセスはC言語で言う構造体:task_struct で管理され、プロセスが生成されるタイミングで確保されている。
* プロセスの生成は常に新しいものを生成しているのではなく、既存のプロセスを複製するかクローンを作成している。
* プロセスが生成されると、「生成」もしくは「新規」といった状態となり、その後「走行可能状態」「走行中状態」「ブロック状態」などに遷移して、最後には「終了」という状態になる。

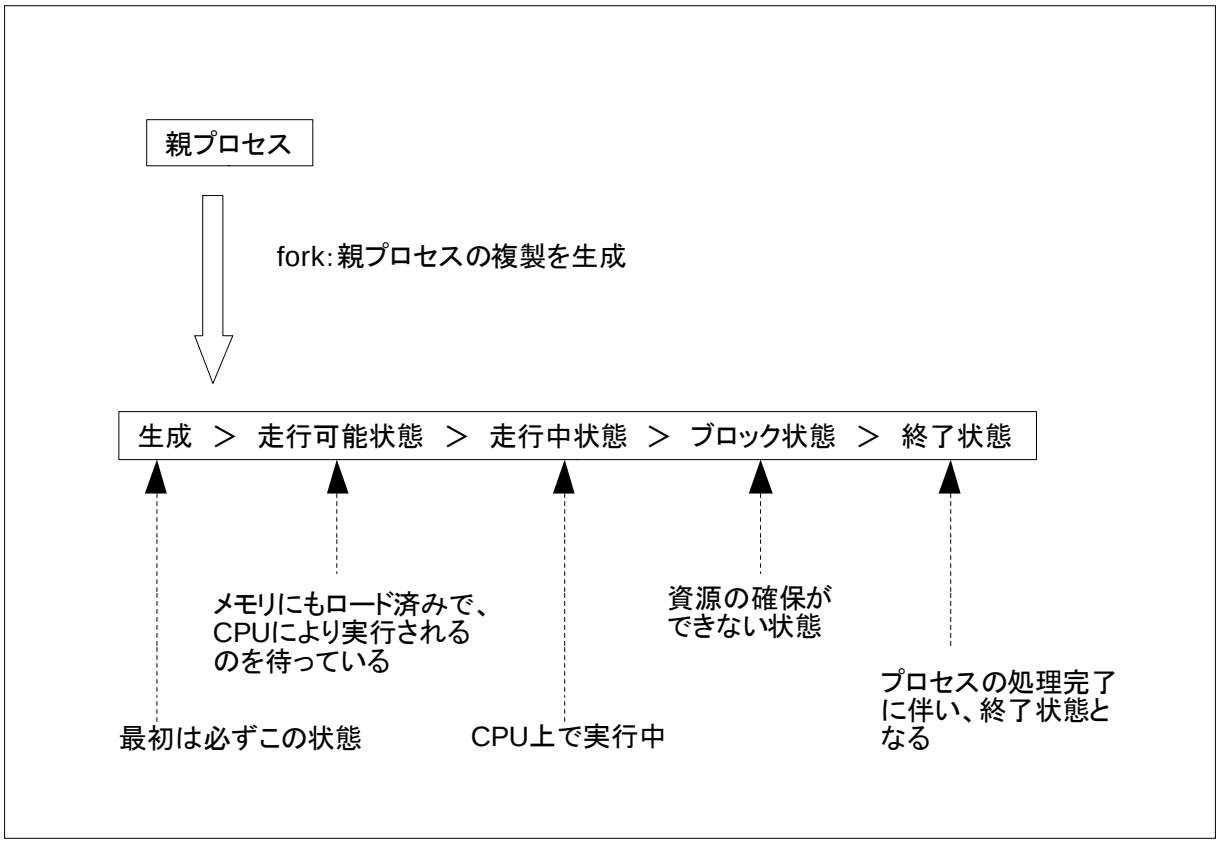

図 I-6-9.　プロセスのライフサイクル

【解説】

1) プロセスの内部データ構造

Linux 上の全てのプロセスはC言語での構造体:task_struct で管理される。プロセスを生成するごとにこの構造体を1つ確保し、以降はこの構造体によりプロセスの管理を行う。以下に task_struct 構造体に含まれる情報を、いくつか紹介する

* メモリ管理構造体 – プログラムに対応する実行命令コードイメージの情報。
* プロセスの優先度 – これをもとに変動プライオリティが計算される。
* 変動プライオリティ – タイムスライスとして使われ、プロセス実行中に減ってゆく。
* コンテキスト – レジスタの内容。
* プロセス ID – 識別番号。
* プロセスの親子関係

2) プロセスを生成する方法

オペレータからのコマンド入力などの指示や、アプリケーションプログラムからの生成命令などによって、プロセスは生成される。ただし実際は、プロセスは常に新しいものが生成されるのではなく、複製されるかまたはクローンが作られているだけである。

* プロセスの生成から実行までは、以下に示すシステムコールを呼び出す。
  - fork() – 親プロセスの複製を生成
  - exec() – 自分自身を新しいプログラムに置き換えて実行

3) プロセスのライフサイクル

プロセスには以下で定義される状態があり、状態遷移はカーネルプロセスによって制御されている。

* プロセスの生成
  プロセスが新規に作成されるとこの状態となり、走行可能状態に遷移されるのを待つ。状態遷移はスケジューラが行う。
* 走行可能(待ち状態)
  メインメモリにロードされ CPU による実行またはリソースの割り当てを待っている状態。
* 走行中
  CPU 上で実際に実行されている状態。プログラムが終了すると終了状態となる。資源が不足した場合はブロック状態にもなる。
* ブロック状態
  資源不足が発生し、かつ新しい資源の確保ができない状態。
* 終了状態
  必要な処理が完了することにより、終了状態となる。

スキル区分	OSS モデルカリキュラムの科目	レベル
システム分野	6　Linux カーネルに関する知識 I	基本
習得ポイント	I-6-10. プロセス間通信	
対応する コースウェア	第4回（システムコール） 第5回（プロセス管理）	

I-6-10. プロセス間通信

プロセスの間でデータやメッセージを通信するための手段として提供されている共有メモリ、セマフォ、メッセージなどを紹介する。

【学習の要点】

* 実行中のプロセスは、専用のアドレス空間が割り当てられるため、他のプロセスのメモリ空間にはアクセスを行うことができない。
* 複数のプロセスの中で、情報の共有などの機能を提供するための仕組みが、プロセス間通信であり、共有メモリや同期、メッセージなどの実装方法がある。

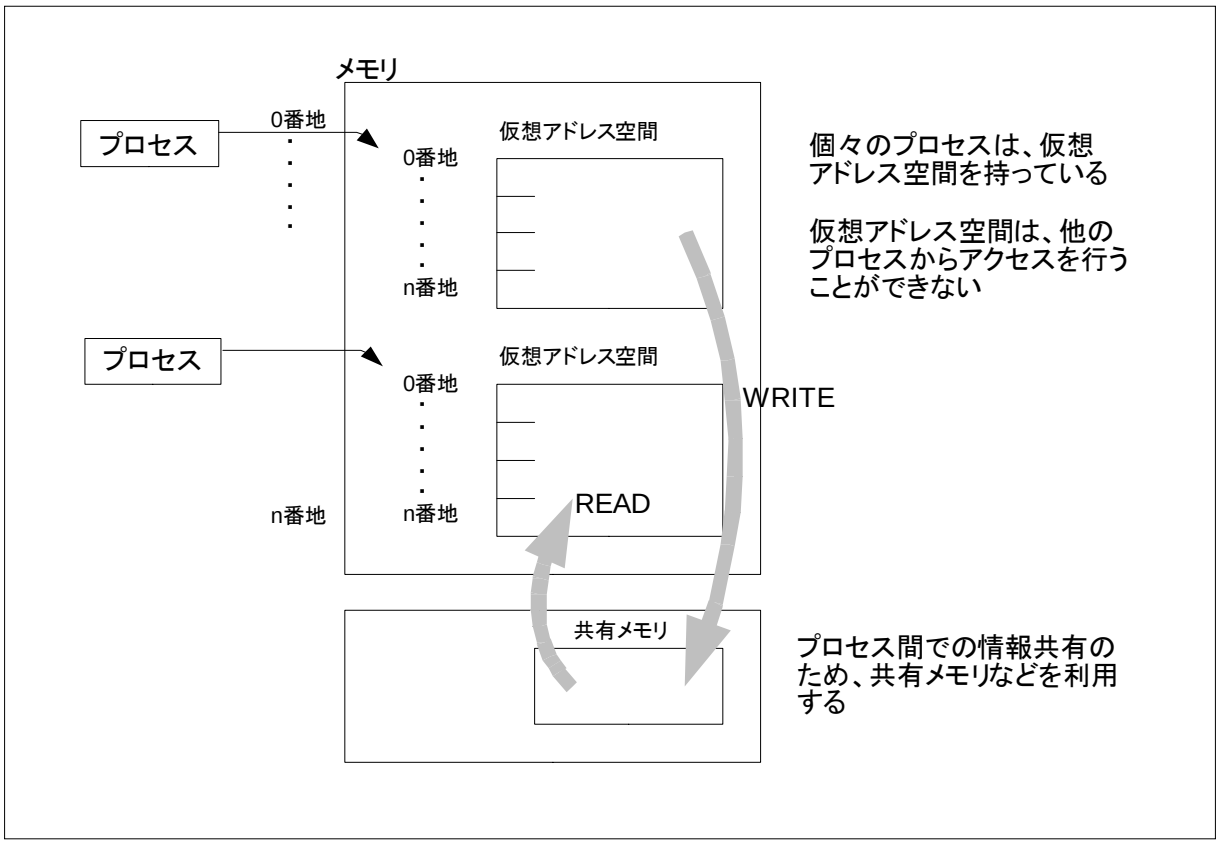

図 I-6-10.　プロセス間でのメモリ共有

【解説】

1) プロセス間での通信

プロセス間通信とは、複数のプロセスの間で情報の共有やメッセージの交換などを行なう機能を提供するための仕組みである。基本的に、実行中のプロセスはプロセスごとに専用のメモリ空間（仮想アドレス空間）が、カーネルにより割り当てられる。これにより、他のプリセスのメモリ空間にはアクセスすることが出来ないため、情報の共有は不可能となる。複数のプロセスで情報の共有を実現させた場合には、プロセス間通信を実装しなければならない。プロセス間通信を実現するためには以下の方法が用いられる。

＊ 共有メモリ

＊ 同期

＊ プロセス間のメッセージ通信、ソケット通信

2) 共有メモリとは

メモリ空間上に共有する領域を確保する方法である。保護機能などは持たないので、動作そのものは高速である。当然、この領域は複数のプロセスからアクセスされるため、整合性を保つためにも何らかの手段を講じなければならない。

3) 同期とは

情報を、どんな場合であっても常に同一の状態と同一の内容に保つことが同期である。これには、情報が保存・格納されている場所などに左右されることはない。同期サービスを有効とするためには、たとえマルチタスクが実装されている状態であっても、リクエストが発生した順番通りに処理を実施しなければならない。同期を実装する手段として、以下がある。

＊ セマフォ
セマフォ変数を複数のプロセスで共有することにより資源の利用が可能かどうかを定める。セマフォ変数は、処理されていないイベント数を表すものである。プロセスは、この変数の示す値が正の整数値であれば1減算して資源を使用し、終了したら 1 を加算する。ただし、0の場合は正の整数値になるまで待機する。

＊ ミューテックス（ロック）
ミューテックスオブジェクトを作成しておき、プロセスが資源を使用したい場合にこれを所有する。もしもミューテックスオブジェクトの状態が、誰かに所有されている状態であれば、非所有状態となるまで待つ。資源の使用が終わったなら、ミューテックスオブジェクトの開放（＝非所有状態）を行う。

4) メッセージ

オブジェクト指向プログラミングにおいて、メッセージ機能によりオブジェクト間での通信が可能となる。これを流用して、情報の共有を実現する。