# ミッション・クリティカル・カーネルとバザール・モデル

## Mission Critical Kernel and Bazaar Model

高 橋 秀 樹

**要 約** 本稿では，まず，ハードウェア技術の発展とともに成長を遂げてきたカーネルの歴史を振り返り，カーネルが進化し続ける必然性を確認する．次に，メインフレーム・カーネルとLinuxカーネルが提供する製品設計思想の違い， 実装方法の違いを検証することにより，メインフレーム・カーネル技術とは何か，Linuxカーネル技術とは何かを探る．更に，両者をミッション・クリティカルという視点で比較することにより，メインフレーム・カーネルが提供する安心感とLinuxカーネルが持つ進取性との間に大きな対立点が存在することを明らかにする．ここには，オープンソースという新しいビジネス商材とバザール・モデルという新しい開発方法論がもたらす大きな課題が横たわっている．ミッション・クリティカルを提供するカーネルには，ユーザに安心感を与えるための機能が必要であり，解析環境に対するカーネル・パラダイムと仮想化技術による新しいカーネル・パラダイムの提供について考察し，結論としている．

**Abstract** In this paper, I confirm first that the continuity of kernels progress is a necessary corollary, reviewing the history of kernels progress accompanied by the progress of hardware technologies. Next, I verify the difference of the product design policy between the mainframe kernel and Linux kernel to find what the mainframe kernel technology is and what Linux kernel technology is. Compared them from a mission critical point of view, I point out that there are several opposing issues between the reliance provided by mainframe kernel and the agility of new technology provided by Linux kernel. Here, we can find big problems brought by new business material of Open Source and new development methodology of the Bazaar model. Kernel that supports mission critical systems must have functions giving reliability to users, and I then conclude with some examinations of providing the kernel paradigm about the analysis environment and new kernel paradigm by virtualization technology.

## 1. はじめに

1990年代半ばに起きたダウンサイジング革命により，メインフレームは恐竜にたとえられいずれは死滅する運命だとされてきた．かつてはメインフレームの強みとされてきたパーティショニング機能やクラスタリング機能はハイエンドサーバ上で実現し，「メインフレームでなければデータセンタを運用できない」という過去の常識は大きく変化しつつある．一方，米国では政府機関や大企業を中心にメインフレーム需要が増加し，電子情報技術産業協会の調査では日本における2004年度の大型メインフレームの出荷台数も前年比24%増を記録し，メインフレームの復権に関する話題も後を絶たない．

本稿では，メインフレームと，オープンソース・ソフトウェア（以下，OSS）の代表格であるLinuxに焦点を当て，カーネル[*1]の進化の歴史，機能比較を通して両者の違いを明らかにする．さらに技術面/ビジネス面の視点から両者の対立点を示し，ミッション・クリティカル・システムを支える上でカーネルに求められるパラダイム[*2]を考察する．

## 2. オペレーティング・システムの歴史

メインフレーム，UNIX，Windows，そして Linux といったオペレーティング・システム(以下，OS）が，なぜ次から次へと誕生し，発展してきたかを学ぶことは，カーネルの役割を理解し，これからのカーネルの進むべき方向性を見極める上で大変に重要な手がかりとなる．本章では，ハードウェア技術の進歩と OS の発展の歴史についてまとめてみる．

### 2.1 OS の歴史的変遷

カーネル技術は，その時代のハードウェア技術の上で成立する．本節では，名機と言われながらも生き残れなかったコンピュータや，誰もその成功を予想できなかった Linux の背景等を踏まえながら，図 1 に沿って OS の歴史的変遷を概観する．

#### 2.1.1 メインフレームの歴史

メインフレームの歴史は，ハードウェア・ベンダの独自 CPU 開発の歴史と言い換えることができる．世界で最初のコンピュータは 1946 年にペンシルバニア大学で発表された ENIAC であるが，ENIAC は単に配線を変更することで実行動作を変えていた．実行動作をプログラム化し，データとともにコンピュータ内部に覚えこませる発想は，1949 年にケンブリッジ大学で開発された EDSAC によって初めて実現した．プログラム内蔵方式の起源は EDSAC である．翌 1950 年には Remington Rand 社（現 Unisys 社）により世界初の商用コンピュータ UNIVAC-1 が誕生している．これらのコンピュータは，まだ OS の機能を持っていなかった．1952 年に開発された IBM 701 は，IOCS（Input Output Control System）と呼ばれる入出力制御プログラムを持っていた．これはユーザ・プログラムからハードウェアを隠蔽するという意味で

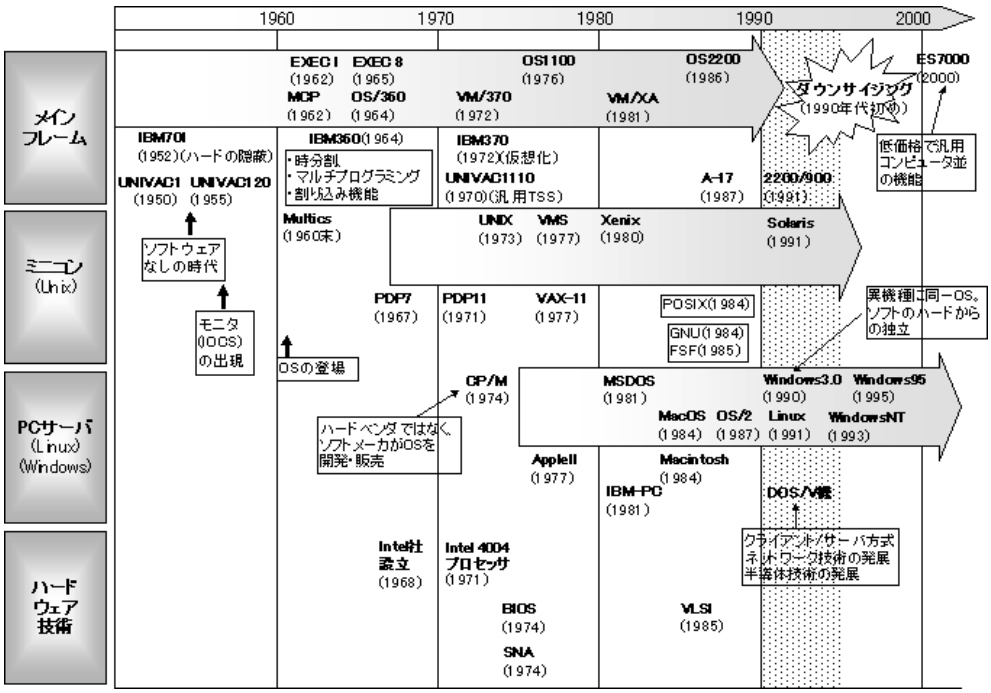


**図 1　コンピュータ発展の歴史**

OS の位置づけにあたるプログラムであり，これを OS の原型と捉えることができる．1962 年には，Burroughs 社（合併後 Unisys 社）が B 5000 にバッチ処理用のプログラム MCP（Master Control Program）を搭載し，本格的な OS 登場の先駆けとなった．1964 年には IBM 360 シリーズに搭載された OS 360 によって，ジョブ管理，タスク管理，ファイル管理，メモリ管理といった OS の基本機能が提供されるようになり，現在の OS 技術は，この OS 360 によって確立されたと考えて良いだろう．ちなみに，旧ユニバック系 UNISYS メインフレーム OS の原型である EXEC 8 は 1965 年に UNIVAC 1108 に採用され，現在の OS 2200 に引き継がれている．1972 年には，IBM 370 が発表され，MVS（Multiple Virtual Storage）や VM 370（Virtual Machine 370）といった OS が仮想記憶環境を提供するようになった．1970 年から 1980 年代はメインフレームとメインフレーム OS の全盛時代であり，1990 年代初頭に起きたダウンサイングに至るまでその存在感は他を圧倒するものであった．

### 2.1.2 Linux 誕生の歴史

Linux は，パソコンをプラットフォームとして開発された OS であるが，UNIX から多くの技術を学んでおり，その意味ではミニコンの制御を起源とする OS である．UNIX は，AT&T が Multics[*3] プロジェクトから降りた後，あるゲームを完成させるために，近くに捨てられていた DEC 社の PDP-7 をプラットフォームとして書いた OS がその原型と言われている．多くの機能を実現しようとする Multics プロジェクトに対して，一つのこと（Uni）に集中する思想から生まれたのが UNIX である．当初はアセンブラ言語で書かれていたが，その後 UNIX を開発するための言語として UNIX 開発者自らが C 言語を開発し，1973 年に C 言語版の UNIX が発表された．UNIX はソースコードが公開され，基本的に無料で使用可能な OS であり，VAX[*4] のオープン OS として発展していく．1984 年に IEEE[*5] によって UNIX ベースの OS 規格 POSIX（Portbale Operating System Interface for UNIX）仕様が制定されたが，その後も様々なメーカが独自に改良を加え，たとえば，Sun Microsystems 社の Solaris，HP 社の HP-UX，IBM 社の AIX，Microsoft 社の Xenix といった多くの独自仕様 UNIX が誕生した．その結果，まったく異なる UNIX が多数存在することになった．

Linux は，このような背景の中でリーナス・トーバルズ氏が 1991 年に個人で一から開発を開始したカーネルである．開発の動機は，UNIX 機よりもずっと安価な Intel 80386 CPU（いわゆる IBM PC/AT 互換機）上で，“ Minix ” よりも優れた Minix を作ることだった．Minix とは，UNIX に似た教育用の OS である．Linux の設計アプローチに関しては，1991 年に「モノリシック・カーネル対マイクロ・カーネル論争」という有名なエピソードが残っている．Linux に関して Minix のニュースグループに意見を求めたリーナスに対して，Minix の開発者であるアンドリュー・タネンバウム教授は，モノリシック・カーネルである Linux を「Linux は時代遅れ」と厳しく評価した．モノリシック・カーネルとは，OS の基本的な機能をすべてカーネルの中に組み込む，いわゆる一枚岩的な方式である．これに対してマイクロ・カーネルは，カーネル機能を必要最低限の機能に抑え，そのほかの必要な機能をカーネル外部に実装する形態であり，カーネルの複雑化/巨大化を防ぐことを目的とする．マイクロ・カーネルを採用する OS としては，Windows NT/2000/XP，Mac OS X，GNU/Hurd[*6] などがあり，モノリシック・カーネルには，メインフレーム OS，UNIX 系 OS，Windows 95/98/Me，そして Linux などがある．その意味では，Linux の実装は，1970 年代のカーネル実装形態から基本的に大

きく変化していない．一方，カーネル機能を小さく分割した場合には，一つ一つの機能は単純化されるが，個々の機能を結びつけるための複雑なコミュニケーションが必要になる．リーナスは，この点に関して逆に Minix における問題点を指摘し対立した．結果的に，時代遅れな実装がエレガントで理想的な実装を押しのけることになったが，この背景には，インターネットによる分散開発環境という革命的な環境変化があった．

Linux は，1997 年頃から商用利用が注目され始め，2000 年頃より IBM 社や Intel 社にフルタイムで雇用されたプログラマも開発に加わるようになった．その結果，ハイエンド・システムに必要な機能が急速に付け加えられていき，2005 年時点では，商用 UNIX とほぼ同等の性能でホストコンピュータとして稼働させることが可能になっている．

### 2.1.3 Windows 誕生の歴史

本稿は Windows の歴史を展開することが目的ではないが，Linux を含めた OS の歴史をまとめる上で，パソコンからの系譜を無視することはできない．1971 年に Intel によってマイクロ・プロセッサが開発されるまでの CPU は，トランジスタを集めた小規模な集積回路で構成されていた．マイクロ・プロセッサは，トランジスタや他の回路素子を一つの大規模集積回路（LSI）として構成したものであり，著しい性能向上と価格低下をもたらした．電卓用のチップとして開発された当初のマイクロ・プロセッサは 4 ビットを扱う機能しか持っていなかったが，2005 年時点では 64 ビットを扱えるまでに発展している．

Windows 誕生までの歴史の中で特筆すべき点としては Digital Research 社が 1974 年に開発した CP/M（Control Program for Microcomputer），1981 年に発表された IBM-PC とその OS として採用された MS-DOS（Microsoft Disk Operating System），更に 1984 年に Apple Macintosh に実装された Mac OS による GUI 機能をあげられるだろう．CP/M は，ハードウェアごとに異なる入出力制御部分を BIOS（Basic Input Output System）として分離し，他の OS 機能はハードウェアに依存しない独立性を保つことを可能にした．Intel チップが多くのハードウェアに対応できたのは，BIOS が開発されたおかげと言っても過言ではない．Mac OS の図形表示，描画機能はパソコン OS の先駆的な役割を果たし，後の Windows OS にも大きな影響を与えた．MS-DOS は，CP/M との互換機能から出発し，多くの UNIX 機能を取り込み，多くのパソコンに採用された．MS-DOS が普及した理由としては，ワープロや表計算といった多くのアプリケーションが Intel x86（80386 等）をプラットフォームとして開発されたことをあげられる．その後，Microsoft 社は IBM 社とともに MS-DOS 後継 OS となる OS/2 を共同開発したが，IBM との共同路線からは離脱し，1990 年に Windows 3.0 を開発，販売した．異機種のパソコンが同一の OS で稼働する形態は Windows 3.0 の大ヒットによって確立され，OS 開発をハードウェア・ベンダの手から切り離したという点でカーネルの歴史における意味は大きい．Microsoft-Intel 連合の成功は，そこから得られる豊富な資金を元にパソコンの更なる性能向上と低価格化を実現し，リーナスの Linux 開発への動機付けを与えることにつながっている．

## 3. カーネルの基本機能

2 章では，ハードウェアの進歩とその時代のニーズが新しいカーネルを生み出し，それぞれのプラットフォームごとに独自の成長を遂げてきたことを見てきたが，本章ではカーネルの役

割とカーネルの基本機能について概説する．

### 3.1 カーネルの役割

人間とコンピュータがコミュニケーションするためには，手となり足となって両者を結びつける橋渡し役が必要である．この役目を担うのが OS というソフトウェアである．OS が持つ機能の中でもシステムの中核となる基本機能を提供する部分をカーネルと呼んでいる．カーネルは特権モードという特別な権限で動作しハードウェアやシステムを制御する．通常のプログラムは非特権モードとして動作するため，ハードウェアやシステムに直接アクセスできないような仕組みになっている．カーネルの役割は，次の一言で表現できる．

「ユーザ・プログラムからハードウェアを隠蔽し，ハードウェアを便利にかつ効率的に操作できる環境を提供すること」

プログラム開発においては，直接カーネル機能を呼び出すのではなく，カーネル機能までも隠蔽するライブラリやパッケージ等を利用することが一般的である．その意味では，カーネルに依存する時代はすでに終わり，プログラミング言語に大きく依存するフレームワーク活用の時代である．しかし，プログラムは，あくまでもカーネルが提供する機能の元でのみ実行可能であり，コンピュータ・システムにおけるカーネルの重要性は今も全く変わっていない．

### 3.2 カーネルの基本構成

汎用的なカーネルは，図 2 のような機能で構成されている．OS の中核となるカーネルには，プロセス管理，メモリ管理，ファイル・システム管理，入出力管理，ネットワーク管理といったシステムを制御するための基本機能が多数割り当てられている．

これらの機能は，実装方法は異なるが基本的にメインフレーム，Linux，UNIX，および Windows 等で共通に提供されている機能であり，各機能はお互いに密接に関係し合いながら機能している．カーネル・モジュールは，必要に応じてカーネルに組み込まれる機能であり，UNISYS メインフレームでは有償機能に対して，Linux ではデバイス・ドライバに対してよく用いられている．

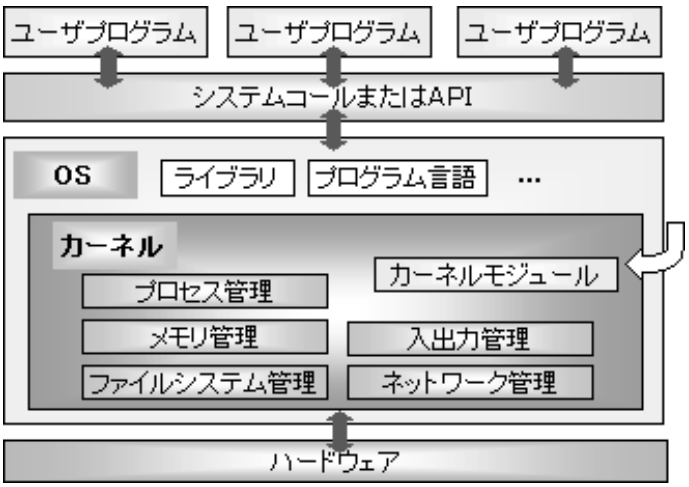


**図 2　カーネルの基本機能**

1) プロセス管理：マルチ・プログラミング等を制御するコンピュータ制御の本質部分
2) メモリ管理：コンピュータのメモリ空間を仮想化する機能を提供
3) ファイル・システム管理：プログラムに対してデバイスを隠蔽する入出力方法を提供
4) 入出力管理：入出力コントローラ制御のための機能を提供
5) ネットワーク管理：年々その役割を増大させている通信制御機能を提供

### 3.3 カーネルの基本機能

3.2 節では汎用的なカーネルが持つ代表的な基本構成を示したが，カーネルの中でも最も基本的な機能はプロセス管理機能（タスク・スケジューリング，プロセス間通信）とメモリ管理機能（仮想メモリ）である．本節では，マイクロ・カーネルを例にとってカーネルの本質となる基本機能について概説したい．

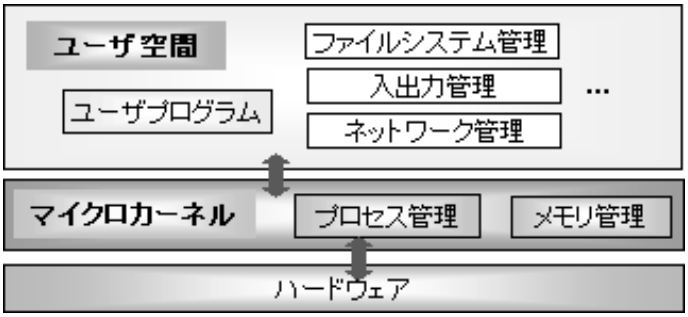


**図 3　マイクロ・カーネル方式の構成イメージ**

図 3 はマイクロ・カーネル方式の構成イメージである．必要最低限の機能だけをカーネル空間に残し，それ以外の機能はユーザ空間に追い出す（アプリケーション化する）ことで，カーネルを簡素化でき，性能も向上するという考え方に基づくカーネル設計方式である．タスク・スケジューリングはカーネルにしか実行できない機能であり，スケジューリング規則の違いはカーネルの特徴を決定付ける大きな要素となる．たとえば，リアルタイム OS は時間制約を守ることが最優先であり，レスポンスタイムを保護することに特化したカーネルと言える．プログラムはメモリ割り当てがなければ実行できないため，仮想メモリを効率的に提供することもカーネルの重要な機能である．一般的にファイル・システムはカーネルの中心機能であるが，これをカーネルの外に出しても機能の提供は可能である．同様に I/O 処理や割込み処理もほとんどの機能をアプリケーション側に実装することが可能である．カーネルを最小限に簡素化することにより，カーネル開発効率の向上，カーネル自体のメモリ占有領域の削減，移植性/保守性の向上，といったことが可能になる．しかし，マイクロ・カーネルの最大の欠点として，細分化された機能コンポーネント間で発生するメッセージ通信負荷があり，これによる処理全体への非効率性が大きな問題となる．カーネルの基本機能は，そのカーネルが提供するパラダイムによって大きく変化するものであり，ここにカーネル設計の面白さと難しさが潜んでいる．

## 4. メインフレームと Linux カーネルの技術比較

本章では，メインフレーム・カーネルと Linux カーネルに焦点を絞り，技術的な観点から両者の違いを比較してみる．

### 4.1 メインフレームと Linux の製品設計における差異

表 1 は，メインフレームと Linux の製品設計における相違点を比較したものである．これをミッション・クリティカル視点で捉え，それぞれの項目に対してミッション・クリティカルの提供に有利と考えられる内容に網掛けをつけて表記している．

**表 1　メインフレームと Linux の製品設計における相違点**

| メインフレーム（UNISYS OS2200） | 比較項目 | Linux |
|---|---|---|
| システム･ダウンを起こさないシステム構築 | ①問題原因の追究度合 | メインフレーム同等機能を追及中 |
| ハード/基本ソフトの閉鎖性 | ②本質的な安心の追求 | ハード/基本ソフトの公開で攻撃要因が発生 |
| 開発･設計･製造･導入･保守の一元化 | ③システム･ライフの保証 | 各社のコアコンピタンスのサービスに依存 |
| 独自 CMOS プロセッサ | ④対応プロセッサ | 多くのアーキテクチャに対応 |
| 独自入出力プロセッサで多重同時処理 | ⑤入出力効率 | 高速バス方式で入出力順次処理 |
| CPU,メモリ,ディスク，ネットワーク等の故障分離, | ⑥縮退運転 | 動的分離に課題 |
| 障害の検出/診断に対するハード機能（99.999%の稼働率確保，RAS 最優先） | ⑦システムとしての設計思想 | メインフレーム同等機能を追及中 |
| 特定責任者による OS 環境の設定･変更 | ⑧システム環境の公開度 | 公開された OS 環境設定 |
| 言語･開発環境を一元/一体開発で提供 | ⑨一元責任/一体化 | ISV 提供言語，開発環境を利用 |
| 運用管理/リソース管理の環境を一元提供 | ⑩運用管理 | メインフレーム同等機能を追及中 |
| 上位互換性を確保 | ⑪顧客アプリケーションの継承度 | 先端機能で差異化優先 |
| トラブル経験を重視した設計･構築 | ⑫システムの堅牢性 | 先端技術の活用による効果を追求 |
| ホット＆ウォームスタートでリカバリ | ⑬リカバリ機能 | 商用クラスタリング製品に依存 |
| 筐体とカーネルが一体の強固なセキュリティ | ⑭セキュリティ機能 | ISV/IHV 提供のセキュリティ製品を利用 |
| バッチ，会話型，トランザクション，リアルタイム等，オールラウンドなホスト集中処理 | ⑮処理形態 | 特定アプリケーションによる分散処理 |

ミッション・クリティカルへの適用という観点でメインフレームと Linux カーネルの設計方針を比較した場合，メインフレーム・カーネルは，元々ミッション・クリティカル・システムを提供するために開発され，提供されていることが鮮明に浮き上がる．これらの多くは歴史的な背景から生まれているものであり，Linux カーネルとの生い立ちの違いをそのまま反映した結果と考えて良いだろう．このような中で，Linux カーネルのミッション・クリティカル対応が声高に叫ばれ，事実として一部のミッション・クリティカル業務に Linux が適用され始めていることを理解するためには，それぞれの設計方針を理解するだけではなく，別な視点から分析を試みる必要がある．

### 4.2 実装から見た技術的相違点

表 2 は，メインフレームと Linux の実装内容を比較したものである．実装という点では，それぞれの開発プラットフォームが異なり，開発言語も異なることから両者の優劣の比較は大変に難しい．しかし，実装内容を確認することで設計思想の実践度合いとある程度までのカーネル品質を確認することができる．

両者の実装を比較した場合，メインフレーム・カーネルは，提供機能にきめの細かさがあり，

**表 2　メインフレームと Linux の実装内容の比較**

| メインフレーム（UNISYS OS2200） | | Linux（カーネル 2.6） |
|---|---|---|
| デバイスをチェック/初期化し，OS をローディングするブートローディング機能は，PC における BIOS ブートローダと大きな差異はない．OS をローディングする前に，必要であればパニック・ダンプの採取処理が行われる点が，Linux とは大きく異なる． | ブートローディングと初期化 | BIOS によりブートローダを起動し，そこから IPL（Initial Program Loader）が起動され，OS 初期化処理に至る機能自体は，メインフレームと同様である． |
| アドレス空間は 2 の 54 乗-1 ワードという巨大空間であり，4096 ワード（バイトではない）を 1 ページとする仮想記憶である．サブシステムと呼ばれる処理単位ごとに 4 ギガワードのアドレス空間が割り当てられる．さらに必要な場合は，4 ギガワード単位で増やすことが可能．メモリ参照頻度に応じて，プログラムごとの未使用ページのまとまり（ワーキングセット）が磁気ディスクにはき出される．これによりプログラムの使用実態に合ったメモリ管理を実現している．カーネル空間とユーザ空間は独立している． | メモリ管理 | CPU アーキテクチャによって異なるが，32 ビット・アーキテクチャでは，プロセスごとに 4 ギガバイトのアドレス空間が割り当てられる．アーキテクチャにより異なるページ・テーブル･エントリ形式をアーキテクチャ依存コードの中に隠蔽することで複数のアドレス変換方式を実装している．4 ギガバイトの仮想空間を 3 ギガバイトのユーザ空間（セグメント）と 1 ギガバイトのカーネルセグメントに分けて使用．カーネル 2.6 から，ユーザ・プログラムは 4 ギガバイトの仮想空間を使用できるようになった． |
| Linux におけるコンテキストは，OS2200 ではアクティビティに相当する．マルチ･プログラミング制御は，ダイナミック・プライオリティ方式（過去の CPU 使用実績を元にプライオリティ変更を行いながら CPU ごとの多重待ち行列をハンドリング）によるスケジューリングを採用している．これにより，リアルタイム処理とバッチ処理の両方に対応している． | マルチ･プログラミング管理 | カーネル 2.4 では，単一のキューに実行可能なプロセスをすべてつなぐ初歩的なスケジューラだったが，カーネル 2.6 にて O(1)（オーダワン）スケジューラが導入され，プライオリティごとのキューが各 CPU ごとに存在し，マルチ・プロセッサ環境にも耐えられるスケジューラに進化した．NUMA 機に対するスケジュール機能も提供されている． |
| 資源の割り当て単位は，バッチ処理，タイム･シェアリング処理といった処理形態ごとに生成されるランと呼ばれる単位であり，PCT（Program Control Table）と呼ばれるテーブルで管理される．プログラムはタスクとして管理されるが，資源管理の単位はランであり，ランが Linux におけるプロセスに相当する．CPU 制御の最小単位（コンテキスト）はアクティビティと呼ばれ，Linux におけるスレッドに相当する． | プロセス/スレッド管理 | 資源割り当ての単位は，プロセスであり，task_struct 構造体により管理されている．CPU 制御の最小単位はスレッドである． |
| コモンバンクと呼ばれる共有メモリやファイルの共有，メッセージ・キューイングにより，ラン間の情報伝達を行う．同期取りは，CPU 間での同時参照に対しては TS 命令やシグナルが使用され，資源操作が絡むような時間のかかる同期取りにはセマフォが使用される． | プロセス間通信/同期 | パイプ機能は，無駄なプロセス切り替えを防ぐためにプリエンプションを抑える実装になっている．ソケット機能，共有メモリ，セマフォ，メッセージ機能（System VIPC）も実装されている．同期取りは，TS 命令，シグナル，セマフォに加え，Read Copy Update という資源排他機構や，同一ページをプロセス間でマップし，システムコールを介さずに排他制御する FUTEX という新しいプロセス間通信機能が提供されている． |
| 一般ユーザの I/O は，ファイル・システムを経由してカーネル内の I/O ハンドラが H/W コマンドを発行する．カーネル内にはデバイスごとにキューイング機構があり，I/O 要求の優先度，ディスク・ポジショニングによる並べ替えが行われる．デバイスへの多重パスを提供するための負荷分散アルゴリズムを実装し，割込みは任意の CPU が処理可能である．デバイスに対してファイル・システムを経由せずに直接 I/O を発行することも可能である．また，特定デバイスに対しては，割込み負荷を回避するためにステータス・ポーリングによる I/O 完了処理を採用している． | 入出力と割り込み | カーネル 2.4 までは，バッファ単位の小さな I/O 処理しか提供されず，大量の I/O 処理に弱点があったが，カーネル 2.6 からは複数ページの大きな単位での I/O が可能となり，非同期 I/O の正式サポート，I/O スケジューリング（Deadline, anticipatory, noop)の導入といった多くの機能改善が行われている．割込みについても，割込み遅延処理が完全に書き換えられ，同時並行処理性能が改善されている． |
| メインフレームと遠隔地端末間でのデータ伝送機能を提供するための API を実装しているのみ．LAN 環境でのネットワーク機能は，カーネル内には取り込まれておらず，通信アプリケーションが提供している． | 通信制御 | 通信プロトコルとして BSD UNIX の実装ベースではなく，Linux 独自に TCP/IP プロトコルスタックを実装し，Ipv6 にも対応している．TCP/IP 標準とは別に，パケットフィルタ，NAT 機能，負荷分散機能等も実装している． |
| ファイルへのアクセスはファイル名で行われる（ハッシュ探索）．ファイルにはデータ型が存在し，必要に応じてカーネル内でデータ型がチェックされる．キャッシュ機構はキャッシュファイルの利用を宣言したファイルにのみ適用される．他サーバの配下に直結された共有ファイルに対して同時更新処理が可能である．重要ファイルについては，カーネルによるディスクの 2 重化やジャーナリング機能が利用される．UNISYS 独自のファイル・システムであり，仮想ファイル・システムは存在しない． | ファイル管理 | ファイルへのアクセスはパス名で行われる．ファイルはすべてバイナリであり，カーネルはデータ型を意識していない．カーネルは空きメモリがあると，ほぼすべてをファイルキャッシュ域として利用する．このため，不慮のシステム停止に備えて Ext3 等のジャーナリングファイルが提供されている．仮想ファイル・システムにより，NFS や様々なファイル・システムをハンドリング可能であり，ユーザはファイル・システムを意識することなく透過的に利用できる．ファイル・システムが豊富なため柔軟性に富むが，安定性に関する実績はこれから． |
| ラン単位で処理サービス量を計測する仕組みがカーネル内に実装されている．（課金サービス） | アカウンティング管理 | 実装されていない． |
| トランザクション処理の高速化のために，専用ファイル・システム（TIP File Control）と専用 API が実装されている． | トランザクション制御 | 実装されていない． |

歴史的に深い配慮が施されたものが多い．一方，Linux カーネルは，先端技術の取り込みが早く，通信制御関連等ではメインフレーム技術よりも進化した内容を実装している．コーディング面では，メインフレームがきっちりとしたコーディング・コンベンションの下で開発されているのに対し，Linux は一貫性に欠けている点を指摘できる．OS 2200 は，アセンブラ言語と Pascal に似た UNISYS 独自仕様の PLUS 言語で書かれているが，C 言語で書かれた Linux とはデータ構造を含め全く異なる実装となっている．どちらもモノリシック実装であり，両者を提供機能面で捉えるならば，それほど大きな相違点はないと判断して良いだろう．

## 4.3 カーネルが提供するパラダイムの違い

技術視点から見た場合のメインフレーム・カーネルと Linux カーネルの違いは，一体どこにあるのだろうか？

### 4.3.1 メインフレーム・カーネル技術とは

4.1 節，4.2 節での議論を踏まえた上で，メインフレーム・カーネル技術とは何であるかを定義してみたい．図 4 に示すように，メインフレーム・システムでは，端末の制御を含め，すべての動作をメインフレーム OS の下で制御している．日中はオンライン業務を行い，夜間はバッチ処理を行う．メインフレームの最大の使命は，これらの業務を滞りなく安全に処理することであり，最大の特徴は，自社のハードウェア，自社のソフトウェアという閉鎖性を最大限に生かしてシステムの堅牢性を確保している点である．ミッション・クリティカル・システムを提供するためのハードウェア技術，ソフトウェア技術，保守サポート技術を有機的に結び付けている OS がメインフレーム OS であり，その基本機能を提供しているのがメインフレーム・カーネルである．脈々と流れるダウンタイム・ゼロの発想，そして一見矛盾するように見えるが，止まらないことよりも早く止めることを優先する技術（システム・ハングを防止するためのホスト縮退機能と業務の継続性確保），独自命令セット（独自チップ）でのハードウェアの最適化，ハードウェアとカーネルが一体となって初めて提供可能となるリカバリ技術，そしてトラブル解析を支える高度なデータ収集技術等，ミッション・クリティカル・システムを安全に機能させることが，メインフレーム・カーネルが提供するカーネル・パラダイムである．

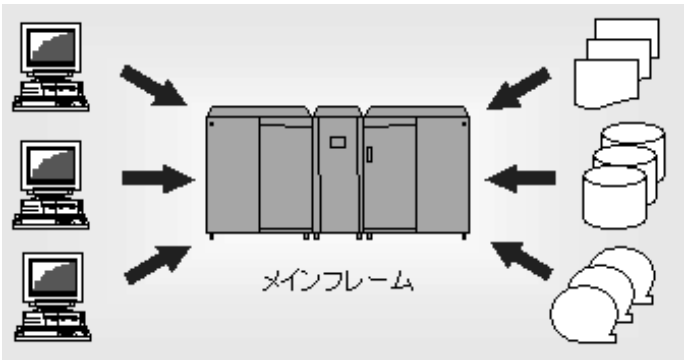


**図 4　メインフレーム処理**

### 4.3.2 メインフレーム・カーネルのセキュリティ

セキュリティ確保を最重要項目とするシステムは，メインフレームが持つ頑強なセキュリティに大きな存在価値を見い出すことができる．UNISYS メインフレームとその OS である OS 2200 の元では，Windows システムで猛威を振るっている Nimda[*7] 等のウイルスが発病できない仕組みが提供されている．その概要を表 3 にまとめてみた．100% のセキュリティを保障することはできないとしても，その頑強さを垣間見ることはできるだろう．

**表 3　UNISYS メインフレーム（OS 2200）におけるセキュリティ**

| ウイルス感染対象 | OS2200 におけるウイルス感染防止内容 |
|---|---|
| カーネルファイル | ディスク上のカーネル領域は，ユーザ領域からは完全に独立し，かつシステム稼働中には変更できない．（オフラインモードでのみ書き換え可能） |
| 共用実行ファイル | Windows の DLL に相当する共用実行ファイルは読込/実行ファイルであり書込みはできない．（特権アカウントによる専用インストーラの起動によってのみ書き換えが可能） |
| ユーザ実行ファイル | デフォルトは読込/実行ファイル属性であり書込みはできない．ユーザ実行ファイルは属性を変更できるためウイルス感染の対象となり得るが，書き換えられたデータが UNISYS の命令語として整合性を維持できない限り，該プログラムは命令語違反でエラー終了する． |
| ディスクのフォーマット | ディスクのフォーマット操作は，ディスクが正常動作中には変更できない．（ディスクのステータスを変更し，特権アカウントによるフォーマット・ツールの操作によってのみ書き換え可能） |
| ファイル管理情報 | ユーザ・プログラムがアクセスできない領域に存在し，かつ管理構造が公開されていないため，ウイルスが直接的にアクセスして書き込むことはできない． |
| ファイル全般 | B1 セキュリティが実装されているため，権限のないアクセスは参照も含めて拒否される． |
| メモリ上への常駐化 | すべてのプログラムは独立した仮想空間を割り当てられ，参照時にアクセスチェックされるため，他のプログラム空間にはアクセスできない． |
| バッファ・オーバフロー | バッファ・オーバフローは UNISYS メインフレームのハードウェア機構によりエラー検知されるため，他のメモリ領域を破壊することはできない．スタック領域は実行不可属性のため，ウイルスプログラムは実行できない． |
| 共有メモリ | カーネル等が使用する共有メモリ領域は，UNISYS メインフレームのリング管理機構によって保護されているため，ウイルスはアクセスできない． |
| 実行中のユーザ・プログラム | プログラムの命令語部分は書込み禁止であり変更はできない．プログラムのデータ語部分は自分自身による書込みのみ可能であり，他ユーザ・プログラムからの書込みはできない． |
| スケジューリング | OS2200 では，あるルールに従うことによってのみプロセス・スケジューリングされるため，単に感染しただけでは発病しない． |

### 4.3.3 Linux カーネル技術とは

Linux における業務システムは，図 5 に示すように 3 層モデル（プレゼンテーション層，アプリケーション層，データベース層）に基づいて構成されることが多い．ほとんどすべてのプログラムは，ユーザ・インタフェースの実行，メインロジックの実行，データの保存/検索という三つの機能を提供する必要がある．メインフレームでは基本的にこれらの機能をメインフレーム本体がすべて引き受ける．

ビジネスアプリケーションがメインフレームからパソコンに移行するようになってからは，まず 2 層モデルとしてクライアント側とサーバ側でそれぞれ分散処理する方法（いわゆるクライアント/サーバ方式）が主流となり，更に導入，改良の容易さ，拡張性，保守性の向上といった観点が加わり，現在では 3 層モデルが主流となっている．

Linux カーネルは，他の UNIX 系 OS と基本構造はほぼ同じであり，AT&T 系の SystemV

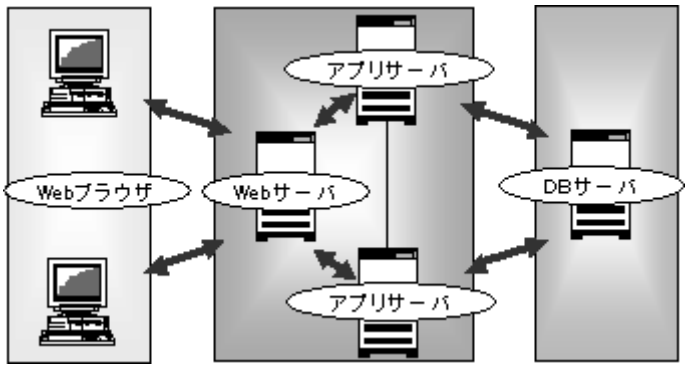


**図5　Linuxシステムの例**

や米カリフォルニア大学バークレー校のBSD（Berkeley Software Distribution）の設計思想を引き継いでいるという点で，技術的な目新しさは見つけにくい．Linuxが普及した最大の理由は，技術的な優位性にはなく，UNIX機能がIntelプラットフォーム上で利用でき，かつそれが一定の安定性を保持していた点にある．600万行に届こうかという巨大カーネルであるが，その約半分はデバイス・ドライバ関連のコードである．

### 4.3.4　Linuxカーネルの拡張性とモノリシック構造

実装に関しては，これまで商用UNIXの性能レベルに追いつくことを一つのターゲットにしてきたと考えられる．Linuxがモノリシック構造を採用した最大の理由は，Linuxの技術的ベースであるUNIXがモノリシック構造であったこと，および，システム効率を優先したことである．マイクロ・カーネルは開発効率，保守性，再利用，検証性等において優れていると考えられているが，実装面ではむしろ多くの欠点を抱えている．プロセス間通信のオーバヘッド（データコピーやコンテキスト・スイッチ）が大きく多量のメモリを必要とする．そのオーバヘッドによる効率低下を抑えるためにカーネルが複雑化し，同等の機能を提供するためのカーネルは，むしろモノリシックよりも大きくなってしまう傾向がある．これらは，マイクロ・カーネルが保守性において必ずしも優位とは言えない点である．一方，Linuxではカーネルが全ての機能を内部に取り込んでいるため，むしろプログラムは単純で書きやすいという側面を持つ．また，ソースレベルでは機種に依存する部分がそれぞれ別のディレクトリに分かれ，概念的に区分されるドライバ類も別々のファイル，別々のディレクトリに分かれている．カーネルの肥大化に対しては，必要な時にカーネルにモジュールを組み込み，必要でなくなったら取り外すという動的な仕組みであるLKM（Loadable Kernel Module）機能が提供されている．現在では性能面を改善した第二世代と言われるマイクロ・カーネルが出現しているが，Linuxはモノリシック・カーネルとして上述のように独自の工夫を取り込んできた．CPUアーキテクチャへの高い移植性もLinuxの大きな利点の一つである．

最先端技術を追い求めるという意味では，メインフレーム・カーネルが技術をリードする役割はすでに終わり，第一線の座をLinuxに譲り渡したと考えるのが妥当である．オープン・スタンダード・カーネルのリーダとしてLinuxが分岐することなく成長を続けていくためには，最先端ITテクノロジへの対応を優先し，既存機能の取り込みだけではなく，真の技術的優位性を強く示していく必要があるだろう．

## 5. メインフレームと Linux の対立点

4 章では技術的な視点からメインフレーム・カーネルと Linux カーネルの違いを見てきたが，本章ではユーザ視点からの比較を行いながら，メインフレームと Linux の対立点について考察してみたい．

### 5.1 ユーザから見たメインフレームと Linux のメリット/デメリット

表 4 にユーザ視点から見たメインフレームと Linux の相違点をまとめてみた．各項目のメリットと考えられる内容に対して網がけの表記をしている．メリットの項目数は，メインフレームが 9 個，Linux が 8 個でほぼ互角である．

**表 4　ユーザ視点から見たメインフレームと Linux の相違点**

| メインフレーム（OS2200） | 項目（開発方法） | Linux |
|---|---|---|
| 有料 | **①ソフトウェア入手料** | 無料（ただし，商用 Linux は有料） |
| 高い | **②導入・構築コスト** | 低い |
| 可能(使用客先のみソース開示) | **③ソースコード解析** | 可能（Web 上でのソース公開） |
| 遅い（自社内資源による開発） | **④開発スピード** | 早い（世界中のコミュニティ・メンバによる開発） |
| 遅い（互換性の維持を最優先） | **⑤新技術の取り込み** | 早い（先端技術の取り込みを最優先） |
| 早い（自社内でクローズ可能） | **⑥トラブル対応スピード** | 遅い（コミュニティを含めた対応が必要） |
| 可能 | **⑦スキル育成** | 可能 |
| 高い | **⑧プロダクト性能** | やや劣る（縮退機能等） |
| 高い | **⑨プロダクト品質** | やや劣る（信頼性，保守性，稼動実績等） |
| 基本的にない | **⑩非互換問題** | 多くの非互換あり |
| 長い（サポート延長が可能） | **⑪サポート期間** | 3〜7 年程度．独力でのサポートなら無期限 |
| あり | **⑫ベンダロックイン** | なし |
| 明確 | **⑬知的財産権** | ライセンスの種類が多く複雑 |
| 自社ハードウェアのみ | **⑭対象ハードウェア** | 多くの H/W に対応 |
| 整備されている | **⑮ドキュメント** | バラバラな情報提供 |
| メーカ | **⑯責任の所在** | ユーザ |
| 多数あり | **⑰対応ミドルウェア** | 少ない |
| 独自仕様が基本 | **⑱開発環境** | グローバル・スタンダード |
| あり | **⑲開発ロードマップ** | なし（OSDL が技術マップを公開している） |

表 4 の内容は，開発方法の違いに着目すると，一つの傾向を浮き上がらせる．Linux のメリットとしてあげられる項目は，その内面にコスト，開発スピード，拡張性（進取性）といったキーワードを含んでいるが，これらはすべてオープンソースの開発手法がもたらすメリットである．一方，メインフレームのメリットとしてあげられる点は，それぞれが高い品質を要求されている項目であり，閉鎖性重視の開発手法がもたらすメリットである．

ユーザはメインフレームの“安心感”をとるのか，Linux の“低価格でオープン・スタンダードな拡張性（進取性）”をとるのか，という大きな対立点の中に置かれている．

### 5.2 開発方法論の相違がもたらす影響

表 4 のメインフレームと Linux における多くの相違点は，その開発方法の違いに根本的な理由を見い出すことができる．

#### 5.2.1 伽藍とバザール

オープンソースの考え方は，「ソースコードを公開して有用な技術を共有することで，世界

中の誰もが自由にソフトウェアの開発に参加することができ，その方が素晴らしいソフトウェアが生まれるはず」という思想に基づいている．1997 年に Eric Raymond が発表した論文“ 伽藍とバザール ”[1]は，Linux という大規模開発の成功を知った作者がその開発手法を自らの OSS プロジェクト fetchmail[*8] に導入し，同様に成功するものかを実験した記録として余りにも有名である．fetchmail プロジェクトは見事に大成功を納め，“ バザール ” モデルの有効性を証明する論文として，未だに OSS 開発におけるバイブル的な存在となっている．バザール・モデルとは，世界に散在する開発者がインターネット環境で開発に参加するモデルであり，Linux が実践してきた開発手法そのものを指している．

Linux の開発スピードは，カーネル 2.6 までのリリースにより実証済みであり，基本的な品質も商用カーネルに匹敵するレベルのものである．また，バザール・モデルには開発スピードの速さ以外にデファクト・スタンダードによるグローバル・スタンダード化の促進という大きな副次効果がある．UNIX は分岐の歴史の中で，本流の UNIX の存在を見失い，多くの異なる商用 UNIX を生んでしまった．バザール・モデルでは，キラーソフトが他を淘汰しながら一つの質の高いソフトウェアを作り上げていく．現時点における機能不足は，短い期間のうちに時間が解決するはずであり，Linux はバザール・モデルという開発手法に支えられ，UNIX 互換カーネルとしての地位を確固たるものにしつつある．

### 5.2.2 Linux の信頼性

表 4 の中でメインフレームのメリットとした項目の多くは，ミッション・クリティカルを実現する上での重要項目であり，Linux は，ベンダ視点（表 1）においてもユーザ視点（表 4）においてもミッション・クリティカルというキーワードの中に弱点を抱えている．しかし，表 1 および表 4 における各項目を技術的な観点に絞って抜き出してみれば分かるように，カーネル・レベルでは両者にそれほど大きな技術上の隔たりはない．ハードウェア自体の信頼性や運用/トラブル対応といったカーネルの外側の信頼性が Linux のミッション・クリティカルへの信頼感を低く抑えている最大の理由と考えられる．

バザール・モデルでは，大規模システム環境で発生するような問題を事前に見つけることが難しく，メインフレームのようにハードウェアからソフトウェアまでを一社のみでカバーするようなワンストップ・サービスの提供も困難である．更に，システムをメンテナンスしていく上での責任の所在は，メーカでもなくコミュニティでもなく，それを利用するユーザ側の責任とされる部分に対して，企業のコミットメントはまだ十分とは言えない．「万一，システムが止まったら？」という問いかけに対して，「1 時間以内に回避策を提示する」といったようなサービス・レベルの提示が可能であるか否かが，ミッション・クリティカル・サポートに対する一つの指標になるが，現在の Linux はこの点に関して大きな課題を抱えている．Linux の信頼性を確保するためには，ベンダ/SIer がその品質確保とサポート責任範囲をこれまで以上に鮮明に打ち出し，バザール・モデルという特性に対しての信頼性技術を強化することが必要となっている．

## 6. ミッション・クリティカルに必要なカーネル・パラダイム

本章では，ミッション・クリティカルを提供する上で Linux カーネルに必要とされるパラダイムを考察してみる．


**図 6　データ解析力の位置付け**

### 6.1　解析環境に対するパラダイムの提供

ソフトウェアの障害解析に関しては学術的な研究が進まない分野であり，学ぶべき教科書も存在していない．先端技術とは距離を置く関係にあり，解析経験がない人にとっては研究対象にしにくい分野でもある．Linux がミッション・クリティカル分野でメインフレームを代替するためには，それを使用するユーザに安心感を与えるためのサポート力を提供できなければならない．

本節では，サポート力を強化する上で大きな要因の一つとなるデータ解析力に着目し，図 6 に示すように，データ解析力をハードウェアとソフトウェアの上に立脚する技術と定義してみる．データ解析力をより具体的に捉えると式 1 のようになる．

データ解析力＝カーネル技術＋解析ツール技術＋個人の解析能力　(式 1)

ここでは，メインフレームの解析環境を参考にすることで，Linux に必要となるカーネル技術と解析ツール技術を考察してみたい．

1)　カーネル技術と解析ツール技術

表 5 は，メインフレームと Linux における解析ツールを比較したものである．Linux における解析環境の現状を分析すると以下のような特徴を指摘できる．

① Linux におけるダンプ取得ツールは，これまでに 6 つのツールが開発されてきたが，結果的にメインフレームの柔軟なダンプ取得機能には追いついていない．

② Linux では，情報をコマンドレベルで個々に取得しなければならないケースが多い．メインフレームのように統合化されたツール/仕組みを用意し，解析における利便性を高めるべきである．

③ Linux では，メモリ上にセーブされる統計情報，トレース情報が少ない．重要な統計情報，トレース情報についてはデフォルトで採取される仕組みが必要である．

④ メインフレームでは，新機能の開発と同時にダンプ編集用コマンドを作成し，ダンプ解析の利便性を維持している．Linux にも同様の文化を育成する必要がある．

障害解析の基本情報は，カーネルがどの情報をどこまで詳細にメモリ上に蓄えるかにかかっている．どのような統計情報/トレース情報が障害解析に有効であるかについては，その情報

**表 5　メインフレームと Linux における解析ツールの比較[2]**

| 機能分類 | OS2200 解析ツール | Linux 解析ツール | 主な相違点 |
|---|---|---|---|
| ダンプ取得 | コンソール・キーインで採取方法を選択<br>①手動採取<br>・磁気テープ装置からカーネル・ローディングし直してダンプ採取.<br>・カーネルのリロードなしにダンプ採取<br>②自動採取<br>・システム停止後自動的にダンプ採取<br>③オンラインダンプ<br>・一瞬動きを完全に停止させダンプ採取<br>・低優先度でなだらかにダンプ採取 | ①diskdump(ハング時ダンプ採取不可)<br>②netdump(ハング時ダンプ採取不可)<br>③LKCD(ハング時ダンプ採取不可)<br>④kdump(システム停止後に採取可能)<br>⑤mkdump(システム停止後に採取可能)<br>⑥LTD(システム停止後に採取可能) | OS2200 では，回復ブート処理の中でダンプが採取される. |
| ダンプ解析 | ①Dumplibs<br>豊富な解析コマンド，カーネル構造への高いアクセシビリティを誇るインタプリタ | ①crash(UNIX SVR 系の解析ツール)<br>②lcrash(LKCD ダンプ用の解析ツール)<br>③Alicia(crash/lcrash 統合解析ツール) | OS2200 の方が圧倒的に解析機能が高い. |
| システム情報取得 | ①コンソール・キーイン(多数)<br>②PAR<br>システム稼動状況の効率分析ツール<br>③LA1100<br>プロセス，メモリ，I/O，ネットワーク，効率等，システム運営に必要となるすべての情報はシステムログ番号で管理されている.<br>(71 番～17650 番) | ①システム全体<br>dmesg,lsmod,lspci,top,ipcs,sysctl,<br>mpstat,sysreport,getinfo,sar,<br>/var/log/messages<br>②プロセス関連<br>ps,pstree<br>③メモリ関連<br>free,vmstat,pmap,<br>④I/O 関連<br>iostat,fdisk,df,lvm,raidtools,mdadm<br>⑤ネットワーク関連<br>ifconfig,ping,netstat,host,route,<br>traceroute,ethool | Linux では，情報取得に対するコマンドの一貫性がない.<br>OS2200 では，統合型ツールによりきめの細かい情報が整理された形で取得可能. |
| トレーサ | ①TR キーイン(カーネル内フックトレース)<br>②IOTRACE(発行 I/O のトレース)<br>③dumplibs(ダンプ内のイベントトレース) | ①プログラムの関数コールトレース<br>strace,ltrace<br>②カーネル内イベントのトレース<br>LKST | OS2200 の方が取得データ量が豊富. |
| プロファイラ | ①LA1100<br>ランログ(プロセス遷移情報) | ①OProfile(カーネル/ユーザ情報)<br>②readprofile(カーネル情報) | OProfile のソースコード参照は優秀. |
| デバッガ | ①PMD/PADS(プログラム・デバッガ)<br>②FLIT(ユーザ用デバッガ)<br>③TFLIT(カーネル用デバッガ)<br>④C キーイン(カーネル更新デバッガ) | ①アプリケーション用<br>gdb<br>②カーネル用<br>kdb,kgdb | OS2200 では，動的にカーネル・コードの変更が可能.<br>Linux は開発中 |

採取に必要となるオーバヘッドを把握した上でカーネルの設計時に検討されていなければならない．解析ツールは，カーネルが提供するインタフェースに基づいて設計されるため，カーネルに対する大幅な変更は既存の解析ツール資産を無駄にしてしまう．ミッション・クリティカル・システムをターゲットとしているカーネルは，カーネル設計の初期段階からデータ解析という観点でのデザイン設計を怠ってはならない.

2)　個人の解析能力

解析ツールを活用するのはあくまでも人間であり，個人の解析能力はデータ解析力を高めるためのキー・ファクタである．ユニアデックスでは，“ ダンプ解析時間の短縮，ダンプ解析スキルの共有・蓄積，カーネル構造解析の逆引き辞典 ” の実現を目指して Linux カーネル・ダンプ解析ツール “ Alicia ”[*9] を開発しオープンソースとして公開している．解析環境を充実させることで，カーネル技術者の負担を減らしながら，データ解析力を高めることができる．解析環境の充実は，ミッション・クリティカルを提供するカーネルが第一に考慮しなければなら

ない重要なパラダイムである．

### 6.2　仮想化技術による新しいカーネル・パラダイムの提供

Linux が，ユーザに真の安心感を与えるためには，非互換を排除し，長期サポートを提供するパラダイムが必要である．互換性を優先すれば新技術の導入を抑制する必要が生まれ，長期サポートを優先すれば開発周期を延ばす必要が生まれる．こうした課題を解消する方法としてカーネルに新たな変更を加えることが更なるカーネルの肥大化を生み，それはコードの複雑化と陳腐化につながる．

カーネルにおける問題点を全く新しい技術で解決するアプローチの一つとして，今，最も注目されている技術に仮想化サーバ技術 Xen がある．Intel 社が，CPU 内に VT（Virtualization Technology）と呼ばれる仮想化支援機能を実装したことにより，今後のカーネル・パラダイムそのものが大きく書き換えられる可能性も出て来ている．VT の目的は，従来，ソフトウェアだけで実現していたソリューション VMM（Virtual Machine Monitor）をハードウェア側で支援することにより，より堅牢で高パフォーマンスの仮想マシンを作れるようにすることである．仮想化技術には多くのメリットがあるが，非互換問題や長期サポート問題を解決するための技術として Xen を利用することも可能である．図 7 に Xen を利用した場合の非互換問題回避イメージを示す．たとえば，新しいカーネル・レベルでは作動しないデバイスを引き続き稼働させたい場合，Xen の制御系ゲスト OS[*10] に対応ドライバがあれば，古いカーネル・レベルの OS を仮想サーバ上に残すだけで OS に修正を適用することなく該当デバイスを使い続けることができる．仮想サーバ技術を利用すれば，カーネル・バージョン間の非互換や古いバージョンの長期サポート問題に対して一つのソリューションを提供できることになる．Xen は，余計な機能を削ぎ落とした一種のマイクロ・カーネルであり，今後のカーネルの発展に大きな影響をもたらす技術である．

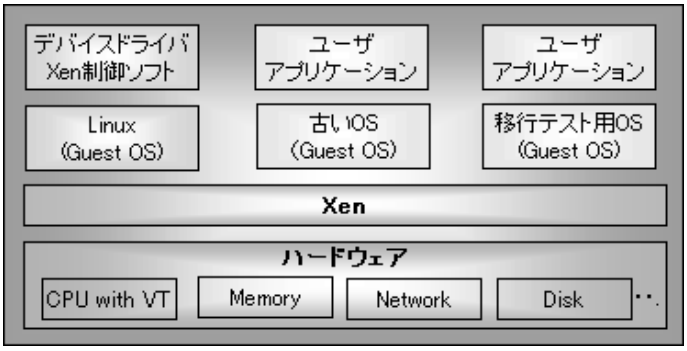


**図 7　Xen による非互換問題等の回避イメージ**

## 7.　お　わ　り　に

世の中はユキビタスの時代に向かっている．たとえば，リアルタイム OS は，今後の新しい方向性を示す OS であり，これまでとは違う小型化されたミッション・クリティカル・パラダイムを提供していくことになるだろう．

すでにミッション・クリティカルを実現し提供しているメインフレームの今後のパラダイムには，オープンプロダクトを積極的にポーティングした上で，頑強なセキュリティを提供して

いくことが含まれるだろう．オープンスタンダードを取り込みながらメインフレームにしかできない機能を提供することが，これからのメインフレーム・カーネルのパラダイムである．一方，Linux もすでに市民権を獲得し，そのパラダイムは，ミッション・クリティカルに向けての技術基盤を提供する方向に向かっている．

米国 Unisys 社においてもビジネス戦略の大きな柱として OSS を掲げており，米国 Unisys 社を含めた日本ユニシスグループの技術は，特に OSS のミッション・クリティカル分野への適用に関して大きく貢献できるものと考えている．ソフトウェアが公共財として広まっていく中で，各企業は独自の生き残り策を考案し，その強みをコミュニティの中でギブ・アンド・テイクしながら進化させていくことになる．その中で，日本ユニシスグループは，“ OSS に信頼感を与える企業 ” になることを目指している．

いずれ，メインフレーム，Linux，UNIX，Windows といったプラットフォームを意識する必要のない時代が来るはずであり，そこでは，カーネル自体の機能よりもアプリケーションを開発するためのプログラミング言語や開発フレームワークが重要視されるだろう．もちろん，アプリケーションの品質/性能がカーネルの特徴/性能に大きく左右されることに変わりはなく，けっしてカーネルの進化が止まることもない．カーネルはその時代のハードウェア技術，ユーザニーズを反映しながら今後も進化を続けるはずである．

カーネル技術者は，カーネルが担うべき最も重要な役割を理解し，技術動向，市場ニーズにマッチするパラダイムを明確に示さなければならない．理論と実装のギャップを見極めながら，最先端技術を理解し適用する能力と情熱が，昔も，今も，そして，これからのカーネル技術者にも強く求められている．

---

＊ 1　オペレーティング・システムの基本機能を提供する中核部分のこと．3.1 節を参照のこと．
＊ 2　ある時代や分野において支配的規範となる「物の見方や捉え方」のこと．
＊ 3　Multiplexed Information and Computing Service の略．1965 年に MIT（マサチューセッツ工科大学）で発足したプロジェクト．
＊ 4　1976 年に PDP-11 の後継機として VMS と言われる OS と共に DEC 社（現 HP 社）によって開発された 32 bit コンピュータ．
＊ 5　The Institute of Electrical and Electronics Engineers, Inc（米国電気電子学会）のこと．非営利の専門機関であり多くの分科会を持つ．主な活動内容は書籍の発行，規格の制定等．
＊ 6　GNU（グヌー）は FSF （Free Software Foundation）が進めている UNIX 互換ソフトウェア群の開発プロジェクトの総称．“ GNU is Not Unix ” という再帰頭字語による名称である．Hurd は GNU プロジェクトが開発中のマイクロ・カーネルの名称．尚，Linux にも “ Linux Is Not UniX ” という再帰頭字語説が存在する．
＊ 7　2001 年 9 月にインターネット上で猛威を振るい大きな被害を出したコンピュータウイルスの一種．
＊ 8　リモートのメールボックスから POP, IMAP などのプロトコルを利用してメールを取得するユーティリティ．
＊ 9　Alicia は，「独立行政法人　情報処理推進機構オープンソースソフトウェア活用基盤整備事業」に係る委託業務の一環として開発した Linux 用クラッシュダンプ解析ツール．
＊10　ハードウェアを動かしている OS を “ ホスト OS ”，その上で動作する OS を “ ゲスト OS ” と呼ぶ．

**参考文献**　［1］　Eric S. Raymond，山形浩生　訳，The Cathedral and the Bazaar(伽藍とバザール)，pp. 28〜60
http：//www.catb.org/~esr/writings/cathedral-bazaar/（原文）
http：//cruel.org/freeware/cathedral.html（翻訳）
［2］　OSS 技術開発・評価コンソーシアム，「OSS 性能・信頼性評価/障害解析ツール開発」OS 層〜障害解析の手順・ツール評価編〜，IPA，2005 年　11 月，pp. 1-7〜1-14

**執筆者紹介**　高　橋　秀　樹　(Hideki Takahashi)

1981 年群馬大学工学部情報工学科卒業．同年日本ユニシス(株)入社．入社以来，一貫して IX 系メインフレーム OS の開発・保守に従事．2003 年からはメインフレーム業務に加え，ES 7000 Linux の受入・開発に従事．2004 年からは OSS 関連業務にも着手し，日本 OSS 推進フォーラムのユニアデックス代表メンバとして参加．現在，ユニアデックス(株)プロダクト事業グループ　ソフトウェアプロダクト統括部 OSS 推進室長．

All Rights Reserved. Copyright® 2006 Nihon Unisys, Ltd.