ライブラリOSをとりまく現況

2.1 まえがき

ここ数年、ライブラリOS（オペレーティングシステム）と呼ばれるソフトウェアが注目されています。本稿ではこのようなソフトウェアの登場の意味と、技術研究所で研究開発をしているソフトウェアについて紹介します。

2.1.1 なぜ今新しいOSが必要なのか

OSは計算機の進化と共に発明され、今日までその性能や利便性は向上し続けてきました。そして現在では利用シーンごとに様々な種類のOSが存在し、ソフトウェアの中核をなしています。このように成熟した技術であるのにもかかわらず、なぜ新しいOSやその仕組みが必要となってくるのでしょうか？

計算機という資源を所有する単位は、複数人から個人へと移り変わってきました。そして、今日の仮想化技術による疑似計算機では、人が利用者の単位ではなくサービスやセッション単位に1つの計算機とOSが所有可能となっています。このような用途の計算機は、一般化されたOSではなく極限まで機能を切り詰めたOSが必要とされています。また、起動に要する時間は数ミリ秒という高速さが求められ、そのOSの稼動時間も数秒など、ある特定の仕事を終えるとそのOSの稼動も終了するというような使い方が想定されています。

また、データセンターで利用されるプログラムやHigh-Performance Computing（HPC）と呼ばれる科学技術計算では、そもそも用途や動作するプログラムが限定されていることから、特定の性能向上を目的として、一般化されたOSではなくある計算処理のみの性能向上を目的としたOSが求められています。

このような機能を実現するために、1990年代に研究開発が行われていたライブラリOSが再考されています。ライブラリOSは、エクソカーネルと呼ばれる仕組みを元にしたOSです。エクソカーネルは、現行の多数のOSが採用しているモノリシックカーネルと呼ばれる仕組みとは一線を画したカーネルです。図-1に示すようにOSの核となる部分のカーネルには必要最小限の機能のみを担当させ、大多数の機能をライブラリOSと呼ばれる部分に組み込むことによって、機能追加をしやすくしたり、再利用可能なソフトウェアを実現することができます。エクソカーネルは、カーネルの規模を最小限に留めることで、柔軟な用途に耐えうるOSの設計を支えています。

次節では、このような特徴を持つソフトウェアの実現例として、世界各地で提案されているライブラリOSの特徴を紹介します。

2.2 関連研究プロジェクト

2.2.1 MirageOS

2010年のUSENIX HotCloudで発表されたMirageOSは、当初Xenハイパーバイザ上で通常のOSの介在なしに動作するOcamlランタイムとして研究開発が進みました。イギリスのケンブリッジ大学の研究者らを中心に研究開発がされています。

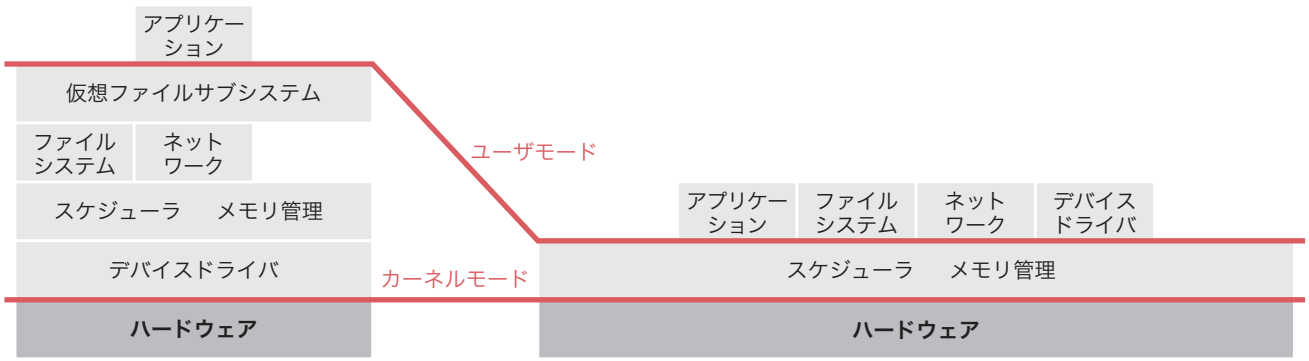

図-1 モノリシックカーネルとエクソカーネルの概念の比較*1

*1　出典元：https://commons.wikimedia.org/wiki/File:OS-structure.svg。

彼等の一番の貢献は、この縮小化されたソフトウェアについて、OcamlやXenといった技術に依存しないUnikernelという概念(名前)を生んだことです[*2]。

2.2.2 DrawBridge

ほぼ同じ時期にマイクロソフト研究所の研究者らによる、DrawBridgeと呼ばれる技術がACM ASPLOSで発表されました[*3]。DrawBridgeはWindowsカーネルのソースコードをユーザ空間が利用できるライブラリとして移植することで、アプリケーション固有のカーネル機能を実現しました。これらの成果は今日のWindowsの仮想化機能として利用されています[*4]。

2.2.3 Rump kernel

Rump kernelは、フィンランドのAntti Kantee氏らにより研究開発がされた、NetBSDカーネルへの機能拡張です。Rump kernelは、Anykernelという仕組みのもと設計されました。Anykernelは、1つのソースコードをモノリシックカーネルとしてだけでなく、ユーザ空間ライブラリや、Unikernelとして再利用可能なものとすることを特徴としています。

この他にも様々なライブラリOSが研究開発されており、ライブラリOSやUnikernelなどの技術への期待は大きなものとなっています。次節では、筆者も開発に参加している、Linux Kernel Library(LKL)と呼ばれる、Linuxカーネルを元にしたライブラリOSについて紹介します。

2.3 Linux Kernel Library

Linux Kernel Library(LKL)は、2007年頃より開発が始まった、Linuxカーネルのソースコードを変形させたライブラリです。Linux上のユーザ空間のプログラムがリンクして利用したり、Windows上のプログラムも利用したりすることができるように、動作環境のOSやハイパーバイザなどの下層部を隠蔽するための抽象層を、元のLinuxカーネルのソースコードに対して導入しています。

2.3.1 LKLはなぜ開発されたのか？

LKLは当初、ディスクイメージにある別OSの上で作成されたファイルシステム上のファイルを操作するために作成されました。Linuxではext4と呼ばれるファイルシステムが利用されていますが、このファイルを別のOS上で操作するには、そのファイルシステム(ext4)を理解できるソフトウェアが必要です。このソフトウェアを実装しさえすればファイル操作ができる、ということですが、この「さえ」はかなり厄介な問題です。それは、1)ファイルシステムはかなり高機能な部品であり、ソースコードの分量が膨大なので、移植コストも大きい、2)一度移植したとしても、日々進化するソフトウェアの追従は更に大変、といった問題があるからです。

OSの仮想化といった技術により、別OSの上でも他のOSを動作させることは可能ですが、その仮想環境のための資源利用量が一般的に大きいため、効率的ではありません。例えば大量の仮想OSインスタンスを同一ハイパーバイザで起動させる際、最小構成のOSでは数秒程度で起動が完了する一方、(Linuxなどの)一般的なOSを起動させた場合では、1インスタンスあたり10秒から20秒程度の時間を要してしまい、俊敏にOSを起動させることが困難となっています[*5]。

こういったケースにおいて、LKLのようなライブラリが威力を発揮します。LKLはカーネルのソースコードを「そのまま」利用できるため、移植のコストに関する懸念は存在しません。また、仮想マシンのような大掛かりな仕組みを利用せずに、プログラム単体にて別OSの環境を構築できるため、機敏な動作も可能です。

この特徴は、前述のRump kernelなどでも実装された、Anykernelの恩恵による所が大きいです。OSの機能のうち一部の機能を利用するという新たな使い方において、再実装の無駄を省きつつ目的の機能を実現できます。一般的に抽象度を上げて再利用性を高めると、ある種のペナルティが発生し、性能劣化や複雑性が上がるといった問題が起こりますが、LKLではこれらのデメリットに目をつぶり抽象度を優先事項として設計されてい

*2 Madhavapeddy et al., "Unikernels: Library operating systems for the cloud." In Proceedings of ACM ASPLOS 2013, ACM, pp.461-472 .
*3 Porter et al., Rethinking the Library OS from the Top Down. In Proceedings of ACM ASPLOS 2011, ACM, pp. 291-304.
*4 黒米 祐馬, Microsoftとライブラリ OSの最前線, IIJlabセミナー, July 2015
*5 金津ら、クラウド環境を指向するライブラリ OS の起動にかかる時間の比較およびその分類, インターネットコンファレンス2016

ます。次節では、どのような抽象化が設計に反映されているかを説明します。

2.3.2 設計、実装

LKLは、Linuxのカーネルソースコードツリーに内包される、ハードウェアのアーキテクチャとして実装されます。本来このアーキテクチャは、異なるプロセッサ(CPU)の違いを吸収するように構成されたものですが、LKLではこの仕組みを利用して、(実際にはCPUの違いを吸収するものではありませんが)動作環境を外部プログラムで制御できるよう、中継者の役割を定義します。こうすることで、あるCPUアーキテクチャ上で動作するように実装されたLinuxカーネルを、ユーザ空間のプログラムや、他OSのプログラムにおいて利用可能としています。

一方でこのように追加された抽象層は、オーバヘッドを生むことが一般的です。例として、下位層のOSで動作しているパケットの送出タイミングを制御するスケジューラと呼ばれる機能は、LKL内部にも存在するため、単一のパケットに対して複数回制御がなされる可能性もあります。LKLのような仮想化技術にはこのような重複した機能の除去などを念頭に置いた適切な設計が必要となります。

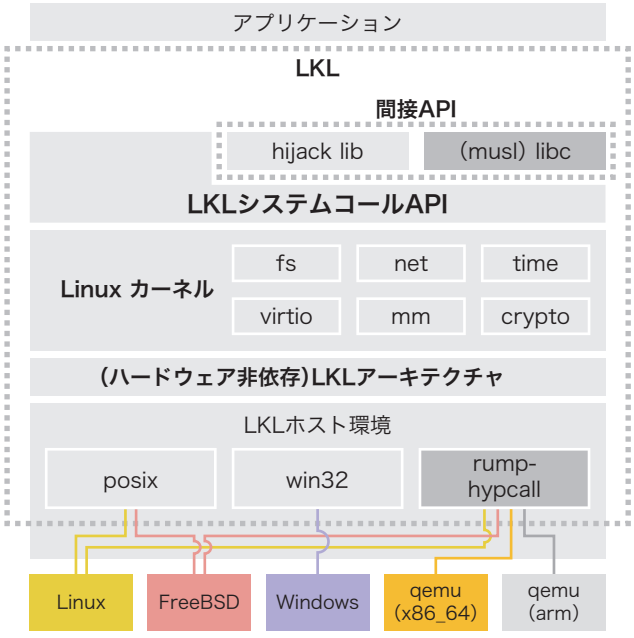

図-2 Linux Kernel Libraryの概要

この中継層は基本的な要件として、時計の管理、プログラムのスケジュール、利用メモリ、の3つの資源を外部プログラムに依存させることが必要になります。この結果、環境非依存な動作を単一のソフトウェアで実現しています。

更に、プログラムがディスクやネットワーク上のデータなど、外部資源とのデータのやりとりをするための入出力機構の中継もこの層で行います。Linuxには、このような仮想的なデバイスを利用するための機構として、virtioと呼ばれる仕組みが用意されており、LKLでもこの仕組みを利用して(再利用して)入出力を実現しています。再利用は単に実装の手間を省くだけでなく、virtioの仕組みを利用した高速化や可用性などの様々な機能を、やはり「そのまま」利用することができる、という面においても貢献しています。

このようにして作成されたプログラムは、ユーザ空間のプログラムがリンクして呼びだすことが可能なライブラリとして作成されます。このままではプログラムは利用しづらいので、アプリケーションインタフェース(API)として、いくつかのものが準備されています。図-2に、LKLの部品の概要を示しました。中心にあるLinuxカーネルのソースコード(と、その実装された機能)を様々な環境やアプリケーションで再利用可能とするために、上下に抽象層が導入されています。現状では、1)LKLシステムコールと呼ばれる、通常のLinuxカーネルが提供するシステムコールのインタフェースと、それを間接的に利用するAPIが存在します(図-2)。間接APIは、2-1)hijackライブラリと呼ばれる、プログラム実行時に関数の書き換えをすることでLKLシステムコールが利用されるものと、2-2)musl libcと呼ばれる標準ライブラリの実装で、プログラムコンパイル時に適切にリンクされるものとが存在します。この後者のLKL用標準ライブラリを利用することで、Unikernelのような用途でも利用できるようになりつつあります。

2.3.3 性能

抽象化・仮想化に伴なうペナルティの度合いがどの程度のものなのかを計測するために、netperfと呼ばれるトラフィック生成プログラムのパケット送信性能を計測することで把握しようと試みました。この計測手法は厳密に言うと完全ではないものの、ソフトウェアの性能について一定の知見を得るための指標となりえます。

計測環境はIntel Core i7-6700というCPU（8コア、3.40GHz）を搭載した2台のPCに、Intel X540-T2という10GbpsのEthernetカード経由で接続された環境で行われました。比較対象にはLKLを利用していない、通常のLinuxカーネル（バージョン4.9.0）上で動作するnetperfプログラムを用いました。

図-3、図-4はそれぞれTCP_STREAM、UDP_STREAMのシナリオにて計測した結果をグラフ化したものです。TCP_STREAM、UDP_STREAMは名前が指すとおり、netperf送信側プログラムが利用するプロトコルをTCPとしたかUDPとしたかの違いです。TCP_STREAMの性能は、netdev1.2[*6]でも報告されているとおり、TCP Segmentation Offload（TSO）やチェックサム計算処理のオフロードがLKLのvirtio-netドライバでも実装されているため、アプリケーションが送信指示するデータのサイズが大きく（65535バイト）、このオフロードが利用可能であった場合には効率的にパケット送出処理が行われ、10%程度のスループット低下に抑えられ性能向上に貢献しています。一方でパケットサイズが小さい場合には現状のLKLの実装ではこのオフロード処理が動作せずに1パケットごとにソフトウェアにてパケット送出処理が行われるため、速度低下の一因となっています。

一方、Linuxカーネルでもオフロードが有効とならないUDPパケットの場合においても、LKLは通常のLinuxカーネルと比べ最大で47%も低い性能でした（パケットサイズ1024バイトのとき）。これは、単一パケットを送出する際に経由する処理が、仮想化によって追加されているためで、改善の余地があることを示しています。しかしながら、アプリケーションが指定するパケットサイズが大きい際には、LKLとLinuxの送出性能がほぼ同等であることが観測されました。これはTCPのときと同様、パケット送出処理がまとめて処理されることによりオーバーヘッドが減少し、結果としてスループット向上に繋がったためと考えられます。

2.4 おわりに

OSは、成熟した技術である一方、新たな用途に応じて形態が増えつづけるため、進化をし続けるソフトウェアの1つです。本稿ではそういった新しい形態のOSが登場してきている背景と、IIJ技術研究所で研究開発をしているソフトウェアを一例に、このようなソフトウェアの存在意義について説明しました。今後のインターネットの発展にむけて、ソフトウェア/オペレーティングシステムのなす役割について再考していただくきっかけになれば幸いです。

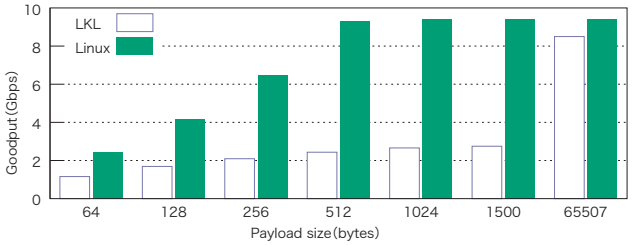

図-3 パケットサイズを変更させたときのTCPパケット転送の性能（Goodput）

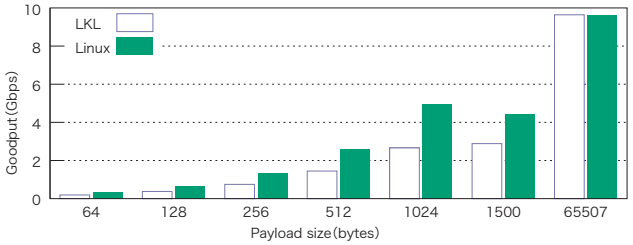

図-4 パケットサイズを変更させたときのUDPパケット転送の性能（Goodput）

執筆者：
田崎 創（たざき はじめ）
株式会社IIJイノベーションインスティテュート 技術研究所 主幹研究員。
インターネットにおけるネットワークアーキテクチャ、ソフトウェアアーキテクチャを専門に研究を行っている。

*6 Chu et al., User Space TCP - Getting LKL Ready for the Prime Time. Linux Netdev 1.2 (Oct. 2016)