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

金津 穂, 田崎 創, 宇夫 陽次朗, 山田 浩史

mkanatsu@asg.cs.tuat.ac.jp,tazaki@iijlab.jp,yuo@iijlab.jp,hiroshiy@cc.tuat.ac.jp

Abstract

クラウド環境において，ライブラリ OS（LibOS）が再度注目を浴びている．LibOS とは OS の機能を有するライブラリであり，アプリケーションとリンクして動作する．クラウド環境では OS 上で単一のアプリケーションのみ稼動させることが多いため，汎用 OS に比べてメモリフットプリントやコードサイズの小さい LibOS が有利となる．またこれらの特徴は早い起動速度をもたらす可能性がある．起動速度はそのクラウドプラットフォームがもつ負荷分散などの *Elasticty* や障害復旧の速度に直結する．しかしながら，これまで提案されている LibOS を起動速度という尺度で比較した調査は著者らの知る限りは無い．本研究では，現行の LibOS の起動にかかる時間に着目し，その定量的な測定および比較，さらにはその測定値を元に LibOS の分類を試みた．

Keywords: クラウド, ライブラリ OS, 計測, Cloud, Library OS, measurement

1. はじめに

クラウド環境のひとつに Infrastructure as a Service（IaaS）プラットフォームが存在する． IaaS プラットフォームでは，ユーザは自身のアプリケーションの負荷に応じて，アプリケーションをホストする仮想マシン（VM）の計算資源量をオンデマンドに増減させることができる．たとえば，負荷が増加した場合には自身の VM（インスタンスと呼ぶ）の起動数を増やし，負荷が減少すればインスタンス数を減少させることで，柔軟に負荷の増減に追従可能である．実際に，Amazon EC2 [1]，Google Compute Engine [2]，Microsoft Azure [3] といった IaaS プラットフォームが実運用されている．

ライブラリ OS（LibOS）とは OS の機能を有するライブラリであり，アプリケーションとリンクして動作する [4]．IaaS プラットフォームでアプリケーションを稼動させる場合，ひとつのアプリケーションや言語ランタイムを動作させる場合が一般的であり，汎用 OS の多くの機能が不必要となる．結果として，その不要な機能のために，メモリフットプリントが大きくなりやすく，プログラムの実行経路も長くなる．LibOS では必要となる OS の機能のライブラリをリンクすればよいため，汎用 OS より軽量であり，メモリフットプリントが小さくなり速度の面でも有利とある．また，構造も簡素になりやすいため，汎用 OS に比べて脆弱性も生じにくい．

LibOS はその軽量さから，早い**起動速度**を有する可能性がある．起動速度は資源利用効率やサービスの可用性に直結する要素である．たとえば，起動速度が十分に早ければ，負荷が増加した際にそのタイミングで起動を開始してもサービスの品質は落とさない．もし速度が遅ければ，予め余計にインスタンスを起動する必要があるため，資源使用率を低くする．また，一度に起動できるインスタンス数が多い場合でも素早く起動が可能であれば，障害時において迅速なサービス復旧が可能となる．同時起動時において速度が遅ければ，サービスの復旧にそれだけ時間を要し，サービスの可用性が低下する．これまでクラウド環境を指向する LibOS が数多く提案されているものの，起動速度という面においてこれらを比較した調査や文献は，著者らの知る限り存在しない．

そこで本研究では LibOS の起動速度に関する定量的な比較をする．LibOS の起動にかかる時間を，クラウドで想定される複数の起動パターンで計測する事により，LibOS の構成法の違いによって起動性能が異なることを明かにし，これを分類する．本稿では提案手法を用いて，4 つの環境について計測を行った．QEMU+KVM のインスタンスを用いて実験を行い，複雑な操作を行わずにそれぞれの環境について分類を試みた．

その結果，以下のことがわかった．

- 理論上最軽量のインスタンス，LibOS 2 種類，Linux の 3 つでそれぞれ起動時間から分類が出来た

Table 1: ライブラリ OS の比較表

	対応 VMM	実装言語	LOC
OSv[5]	KVM/Xen/VirtualBox	C++	367997
IncludeOS [6]	KVM	C++	15486
MirageOS [7]	Xen	OCaml	9075
Alpine Linux [8]	KVM/Xen/VirtualBox	C	16047890

- それぞれの起動時間について，メモリの消費量との相関が見られた
- LibOS の実装の違いについては起動時間から断定できるほどのデータが見られなかった．

2. LibOS の分類

2.1. *LibOS* の構成

LibOS とは，OS の機能を有するライブラリである． LibOS は動作する環境で 3 つに大別する事が可能である．

ベアメタル 最初期の LibOS である ExoKernel [4] はベアメタルマシンの上で動作する LibOS であった．現在この形でしか動作しないものは少ないが，他の構成でもベアメタルでも動作するものは存在する [9].

プロセスベース ホスト OS のプロセスコンテナとして動作する ライブラリ OS．中でも軽量を指向したものとしてピコプロセスが存在する [10, 11].

ハイパーバイザベース ハイパーバイザの上で動作することを前提とした設計の ライブラリ OS.

IaaS プラットフォームで用いられる LibOS はハイパーバイザベースのものである． 今後，本稿ではこのタイプの LibOS のみを対象とする．

2.2. *LibOS* の比較

ハイパーバイザベースの LibOS について，いくつかの代表例について比較表を Table 1 に示す．

3. 関連研究

既存の ライブラリ OS を多数起動した場合の起動速度計測を行った研究として，ClickOS [12] が挙げられる． 彼等の論文では，軽量化された VM がいかに早くユーザのリクエストに応じて起動できるかを示すために，連続的に起動させた時の VM そのものの初期化時間と，その VM で動作するアプリケーションの初期化時間とを分けて計測した．この結果により，ClickOS が Xen のボトルネックを検出した．

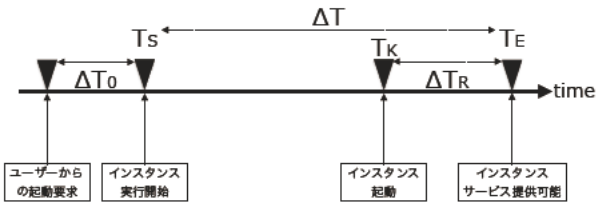

Figure 1: 計測におけるブートシーケンスと用語の定義

Williams 等による Unikernel Monitor [13] においてもゲスト OS が起動してから仮想コンソール，仮想ネットワークデバイスに出力がされるまでの時間を測定した． ゲスト OS が Unikernel のような特定の機能しか利用しないものの場合においてハイパーバイザの処理をどのようなオーバーヘッドがあるかを，最小構成のハイパーバイザー ukvm を実装し比較実験をした．

本研究では，単一のライブラリ OS についてではなく，複数のライブラリ OS と軽量 OS を比較することにより， ライブラリ OS のコア実装における差を起動にかかる時間という尺度で比較できることを明かにした点で，ハイパーバイザのオーバーヘッドを明らかにした両研究と異なる．

4. 起動にかかる時間の計測による分類手法の提案

4.1. 起動時間

IaaS プラットフォームでは，インスタンスの起動速度がサービスの応答速度に直結する．もし起動性能の低い LibOS でサービス品質を損なわずにシステムを運用しようとした場合，インスタンスをホットスタンバイさせなければならず待機インスタンスのリソースに無駄が発生する．

LibOS には様々な構成があり，同じハイパーバイザベースのものであっても起動にかかる時間は異なる．そこで，起動にかかる時間を（1）単一インスタンスの場合（2）複数インスタンスの場合でそれぞれ計測を行い，比較する事で， LibOS の構成の違いによる特徴を明かにする．

OS における起動時間はどの時点をもって起動開始／起動終了とするのか，複数の場合が考えられる．

本研究では起動の開始時刻，起動にかかる時間，起動の終了時刻をそれぞれ T_s，ΔT，T_e と置き，その定義を Figure 1 に示す時刻と時間とした．

T_S 起動開始時刻を表す．計測対象のプログラムの，プロセスが起動する直前の時刻を起動開始とする．

T_E 起動終了時刻を表す．計測対象のプログラムの標準出力から，起動完了を示すメッセージを計測プログラムが受け取った時刻を起動終了とする．

ΔT 起動にかかった時間を表す．T_s と T_e の差分を起動にかかった時間とする．

T_K 計測対象のプログラムの，ホスト OS と VMM と QEMU の初期化にかかる時間を除いた起動開始時刻

ΔT_R 計測対象プログラムの，カーネルの起動後，サービスの起動まで終了した時刻

今回の計測では，OS の内部構造を調べ，手を加える必要がある T_K ならびに ΔT_R の計測は行なっていない．ΔT_R について，LibOS は OS がアプリケーションとリンクしているためほぼ 0 になると考えられるが，Linux のような一般的な OS ではカーネルの起動後に init システムやサービスの起動があるため，その場合は 0 にはならない．

4.2. 計測の手法

計測について，次の 2 つで行った．

- 単一インスタンスの場合
- 複数インスタンスの場合

それぞれの場合について以下に説明する．

単一インスタンス 計測対象を 1 台のみ起動し，T_s，T_e，ΔT についてそれぞれ計測をする．

複数インスタンス 複数のインスタンスを起動し，それぞれのインスタンスについて T_s，T_e，ΔT をそれぞれ計測する．複数のインスタンスの起動方法として，各インスタンスを間隔を空けてシーケンシャルに起動を行なった．今回は，3 ミリ秒，5 ミリ秒，7 ミリ秒の場合で計測した．

5. 計測プログラムの実装

QEMU+KVM [14, 15] で構成された LibOS インスタンスの起動にかかる時間を，C/C++ のプログラムを用いて計測した．

計測環境のホスト OS および エミュレータ，VMM への変更を行わずに計測を行うため，次のような構成にした

- 計測プログラムは，複数スレッドを生成し，各スレッドが `fork()`，`execp()` を行う事によってインスタンスを起動する．

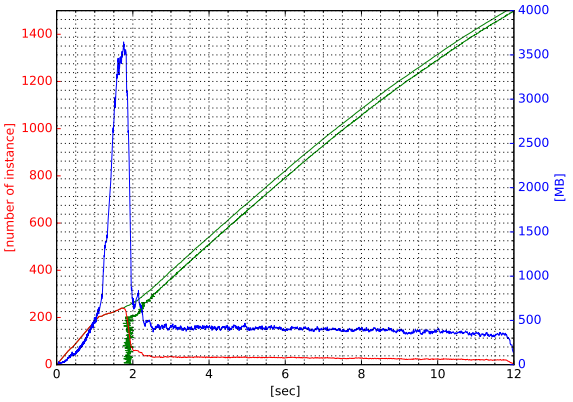

Figure 2: minkernel を 1500 台起動した際の起動時間，同時起動数，消費メモリ量

- 生成されたスレッドは，起動したインスタンスの標準出力を監視し，起動メッセージの出力を観測する事によって起動の終了時刻とし，これを記録する．

また，QEMU には次のオプションを与えた

```
-nographic -enable-kvm -cpu host,
+x2apic -m 50 -netdev user,
id=un0, net=192.168.122.0/24,
host=192.168.122.1 -device
virtio-net-pci,netdev=un0 -device
virtio-rng-pci
```

6. 評価

本研究で提案する計測手法を用いて，現状利用が可能な LibOS の実装を，クラウド環境での実運用で想定される大量の VM をシーケンシャルに起動させた場合を想定し，起動に要する時間を測定する．同様の計測を LibOS 以外のいくかの OS でも行い，比較する事により，LibOS の実装の違いによる特徴を明かにする．

6.1. 実験環境

Table 2 に実験に用いた環境を示す．今回は VMM として，Google Compute Engine でも用いられている KVM を用いた．

6.2. ベースラインの評価

理論上最軽量のインスタンスを計測環境で動かすことによって，本計測環境のベースライン（最速実行速度）を評価した．軽量インスタンスとしては独自に x86_64 アセンブリ言語で記述した *minkernel* を

Table 2: 実験環境

ベンダー	QCT D51U
CPU	Intel®Xeon®CPU E5-2640 v3 @ 2.60GHz （8 コア 2way）
主記憶	DD4 256GB
ストレージ	Intel S3610 SSD （128GB）
ホスト OS	CentOS 7 （Linux kernel 3.10.0-327.el7.x86_64）
VMM	QEMU（version 2.0.0） + KVM

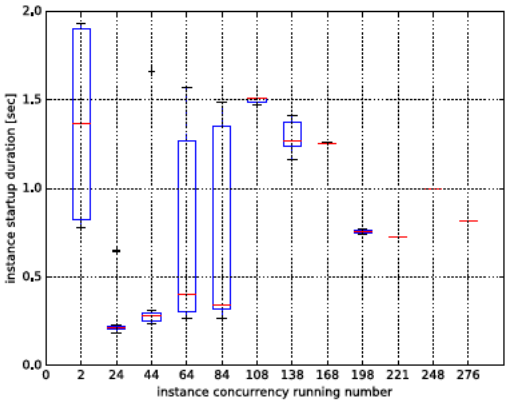

Figure 3: minkernel の同時起動数 N における起動にかかる時間についての最小／最大／中央値の様子

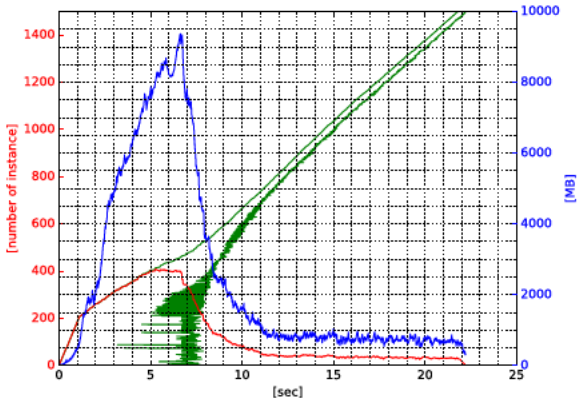

Figure 4: OSᵛを 1500 台起動した際の起動時間，同時起動数，消費メモリ量

用いた． *minkernel* は QEMU 上での動作を前提としており，起動時に以下の 3 処理のみを行う．

- スタック初期化
- I/O ポート（0x3F8）に対する値の出力
- halt

minkernel のブートは QEMU のカーネルダイレクトブートを用いたため，起動にかかるオーバーヘッドは QEMU 上で最小である．

minkernel を用いた計測結果及び同時起動数あたりの実行時間のグラフを Figure 2，Figure 3 に示す．

Figure 2 において，青線はその瞬間でのシステムが使用しているメモリ総量，赤線は同時 OS 起動インスタンス数，緑線の左側は起動開始指示を出した時刻，緑線の右側は起動終了時刻を示している．以降の Figure 4 においても，同様の凡例を使用する．

これらの図から以下の事実がわかる．

- *minkernel* の実行にかかる時間は極めて小さく，1 インスタンスあたり平均 200 ミリ秒で実行が終了している
- *minkernel* の実行に必要なメモリ量は少ないため，同時実行数が増えた場合でも計測環境のメモリ使用量に対する影響がない
- Figure 2 では 5 ミリ秒ごとにインスタンスを起動しているが，ホスト側の初期起動処理（ディスク読取など）が終わったあと（この図の場合は 2 秒後）には平衡状態になり，40 インスタンス / 秒で処理が行われる．

このように *minkernel* を用いることにより，本計測環境（Table 2）のベースライン値を次のように推定できる．

- QEMU + KVM を用いた場合のインスタンスの起動にかかる時間は最短で 200 ミリ秒前後
- QEMU + KVM のインスタンス 1 つあたり最低でも 13 MB のメモリを消費する

6.3. ライブラリ *OS*

クラウド環境を指向する，ハイパーバイザベースの LibOS は複数存在するが，今回は計測対象として OSᵛと IncludeOS を選択した．これらを選択した理由は次の通りである．

- 既存の汎用 OS のコードを用いる Rump kernel [9] と異なり，一から設計されている

Table 3: OSvと IncldeOS における違い

	OSv	IncludeOS
バイナリ方式	ダイナミックリンク	スタティックリンク
ファイルシステム	ZFS	RamFS
使用ライブラリ	STL + Boost	STL
提供する標準 C ライブラリ	musl libc [16]	newlib [17]
OS の中のユーティリティ	含む	含まない
実行アプリの違いによる，アプリ起動までの実行経路の違い	ある	ない

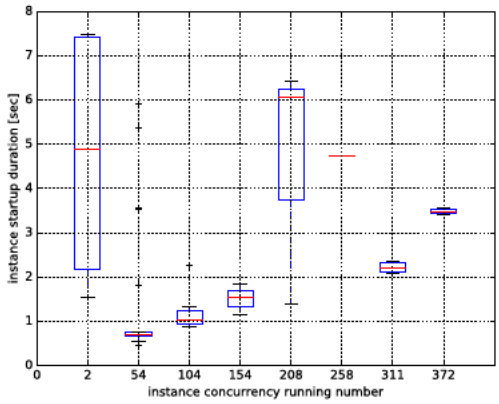

Figure 5: OSvの同時起動数 N における起動にかかる時間についての最小／最大／中央値の様子

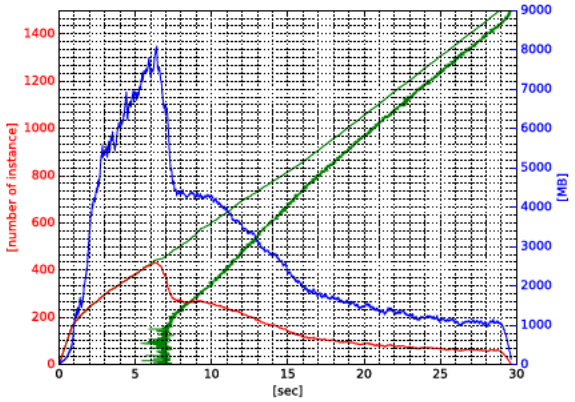

Figure 6: IncludeOS を 1500 台起動した際の起動時間，同時起動数，消費メモリ量

- クラウド環境のハイパーバイザ上で高速に動作させる事を期待して設計された．
- 計測プログラムが前提とする QEMU + KVM の環境で動作する

OSvと IncludeOS はどちらも C++ によって記述され，単一のメモリ空間しか持たない．また，両者ともにユーザプログラムのために標準 C ライブラリを提供する．しかし，ダイナミックリンクを活用し，ZFS のようなファイルシステムを持つ OSvと異なり，IncludeOS の場合は RamFS を使用しており，また，バイナリを全てスタティックリンクにする（Table 3）．

OSvを用いた計測結果及び同時起動数あたりの実行時間のグラフを Figure 4，Figure 5 に，IncludeOS を用いた計測結果及び同時起動数あたりの実行時間のグラフを Figure 6，Figure 7 にそれぞれ示す．これらの図から以下の事実がわかる

- *minkernel* と比較しインスタンス起動にかかる時間が 2–10 秒と大きい
- OSvのほうが起動にかかる時間が平均は小さいものの，分散が大きく安定しない

6.4. *Linux*

Alpine Linux は Linux ではあるものの，設定によってカーネルサイズが小さくされており，ユーザーランドにも musl libc と BusyBox [18] を用いることにより軽量化を行ったディストリビューションである．例えば，Docker のベースイメージとして 5 MB 程度のイメージが配布されている [19]．

Alpine Linux を用いた計測結果及び同時起動数あたりの実行時間のグラフを Figure 8，Figure 9 に示す．Linux の場合 multi user mode と single user mode の 2 つの起動の種類があるが，今回は multi user mode の場合で計測を行った．

グラフより，次の事がわかる

- 単体での起動だと OSvや IncludeOS と大きな差が出ていないものの，複数起動すると顕著に遅い

6.5. 計測結果における考察

どの OS の場合でも，起動を開始したインスタンスの順番が若いものは起動にかかる時間が多く，加えてメモリの消費量と同時起動数も増加していき，ある時点でメモリ消費量と同時起動数が減りはじめ，起動

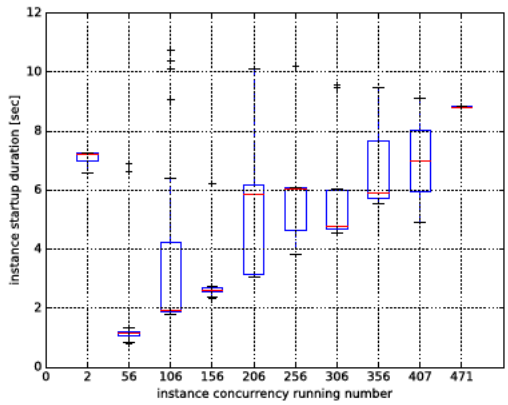

Figure 7: IncludeOS の同時起動数 N における起動にかかる時間についての最小／最大／中央値の様子

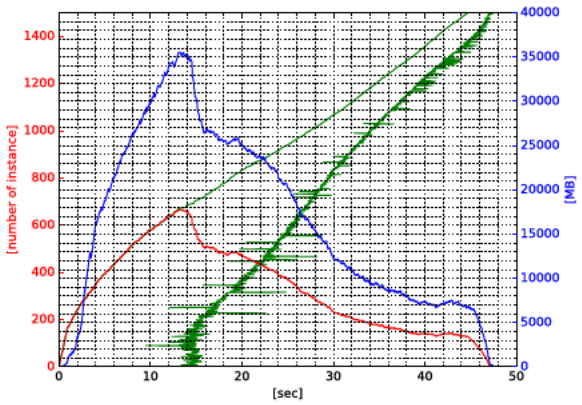

Figure 8: Linux を 1500 台起動した際の起動時間，同時起動数，消費メモリ量

にかかる時間も一定のものになっていくという状況は同じだった．しかし，*minkernel*，OSᵛと IncludeOS の LibOS，Alpine Linux と，この 3 つで大きく起動にかかる時間が異なっている事が数値として明らかになった．

それぞれのメモリ消費の最大値について，*minkernel* が 3647 MB，OSᵛが 9374 MB，IncludeOS が 8091 MB，Linux が 35549 MB となっており，起動にかかる時間が長いものほどメモリの消費量も大きくなっている．OS の起動にかかる時間のボトルネックとして，一般に外部との I/O とデバイスの初期化が挙げられるが，ハイパーバイザベースの LibOS の場合は特定の VMM での 動作に限定し設計することで，外部との I/O 及びデバイス初期化がほぼ存在しない．そのため，メモリの消費量と起動処理の時間が比例する傾向が高い．また，Linux についても，今回

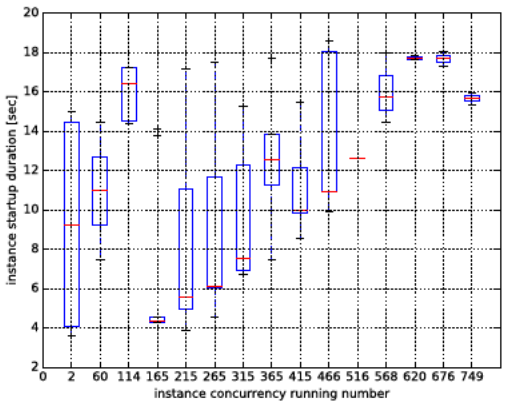

Figure 9: Linux の同時起動数 N における起動にかかる時間についての最小／最大／中央値の様子

は仮想環境上で KVM を用いて動作させたため，同様の傾向を示したと考えられる．

7. まとめ

本稿では，LibOS を想定用途別の起動方法で起動し，その起動時間を計測することで，LibOS の分類を行った．

仮想環境上で複数の OS の起動にかかる時間を観測することにより，*minkernel* と LibOS と Linux については，それぞれ起動の時間に大きな差が見られたが，LibOS の中でも OSᵛと IncludeOS の差については，現在の計測結果からコードパスの違いを断定する事ができなかった．

また，同じ同時実行数の場合でも起動にかかる時間に揺らぎがあり，グラフの誤差が多くみられた．

今後の課題として，システムコールの発行量やかかる時間などの実行プロファイルを，それぞれの OS において調査することで，各 LibOS の実装の差がその結果から分類できないかを調査する．加えて，その結果を本研究の手法にフィードバックし，簡易な時間計測のみでより精度の高い分類を行いたい．

8. 参考文献

[1] Amazon Elastic Compute Cloud, Accessed 2016-08-12. URL https://aws.amazon.com/ec2

[2] Google Compute Engine, Accessed 2016-08-12. URL https://cloud.google.com/compute

[3] Microsoft Azure, Accessed 2016-08-12. URL https://azure.microsoft.com

[4] D. R. Engler, M. F. Kaaspoek, J. O'Toole Jr, ExoKernel: An Operating System Architecture for Application-Level Resource Management, in: 15th ACM Symposium on Operating Systems Principles (SOSP '95), ACM, Copper Mountain, CO, USA, 1995, pp. 251–266. doi:10.1145/224056.224076.

URL https://pdos.csail.mit.edu/6.828/2008/readings/engler95exokernel.pdf

[5] A. Kivity, D. Laor, G. Costa, P. Enberg, N. Har'El, D. Marti, V. Zolotarov, OSv—Optimizing the Operating System for Virtual Machines, in: 2014 USENIX Annual Technical Conference (ATC '14), USENIX Assocation, Philadelphia, CA, USA, 2014, pp. 61–72.
URL https://www.usenix.org/system/files/conference/atc14/atc14-paper-kivity.pdf

[6] A. Bratterud, A.-A. Walla, H. Haugerud, P. E. Engelstad, K. Begnum, IncludeOS: A Minimal, Resource Efficient Unikernel for Cloud Services, in: 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom 2015), IEEE, Vancouver, BC, Canada, 2015, pp. 250–257. doi:10.1109/CloudCom.2015.89.
URL http://dx.doi.org/10.1109/CloudCom.2015.89

[7] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, J. Crowcroft, Unikernels: Library Operating Systems for the Cloud, in: 18th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '13), ACM, Houston, TX, USA, 2013, pp. 461–472. doi:10.1145/2451116.2451167.
URL http://dl.acm.org/citation.cfm?doid=2451116.2451167

[8] Alpine Linux, Accessed 2016-08-19.
URL https://www.alpinelinux.org

[9] A. Kantee, Flexible Operating System Internals: The Design and Implementation of the Anykernel and Rump Kernels, Phd thesis, Aalto University (2012).
URL http://urn.fi/URN:ISBN:978-952-60-4917-5

[10] A. Baumann, D. Lee, P. Fonseca, L. Glendenning, J. R. Lorch, B. Bond, R. Olinsky, G. C. Hunt, Composing OS Extensions Safely and Efficiently with Bascule, in: 8th ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys '13), ACM, Prague, Czech Republic, 2013, pp. 239–252. doi:10.1145/2465351.2465375.
URL http://dl.acm.org/citation.cfm?doid=2465351.2465375

[11] C.-C. Tsai, K. S. Arora, N. Bandi, B. Jain, W. Jannen, J. John, H. A. Kalodner, V. Kulkarni, D. Oliveira, D. E. Porter, Cooperation and Security Isolation of Library OSes for Multi-process Applications, in: 9th ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys '14), ACM, Amsterdam, The Netherlands, 2014, pp. 9:1–9:14. doi:10.1145/2592798.2592812.
URL http://dl.acm.org/citation.cfm?doid=2592798.2592812

[12] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, F. Huici, ClickOS and the Art of Network Function Virtualization, in: 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI '14), USENIX Assocation, Seattle, WA, USA, 2014, pp. 459–473.
URL https://www.usenix.org/system/files/conference/nsdi14/nsdi14-paper-martins.pdf

[13] D. Williams, R. Koller, Unikernel Monitors: Extending Minimalism Outside of the Box, in: 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16), USENIX Assocation, Denver, CO, USA, 2016.
URL https://www.usenix.org/conference/hotcloud16/workshop-program/presentation/williams

[14] F. Bellard, QEMU, a Fast and Portable Dynamic Translator, in: Annual Conference on USENIX Annual Technical Conference (ATEC '05), USENIX Assocation, Anaheim, CA, USA, 2005, pp. 41–46.
URL https://www.usenix.org/legacy/publications/library/proceedings/usenix05/tech/freenix/bellard.html

[15] A. Kivity, Y. Kamay, D. Laor, U. Lublin, A. Liguori, kvm: the Linux Virtual Machine Monitor, in: Ottawa Linux Symposium, The Linux Foundation, Ottawa, ON, Canada, 2007, pp. 225–230. doi:10.1186/gb-2008-9-1-r8.
URL https://www.kernel.org/doc/mirror/ols2007v1.pdf#page=225

[16] musl libc, Accessed 2016-08-19.
URL https://www.musl-libc.org

[17] The Newlib Homepage, Accessed 2016-08-20.
URL http://www.sourceware.org/newlib/

[18] BusyBox, Accessed 2016-08-19.
URL https://busybox.net

[19] Docker Hub - alpine, Accessed 2016-08-20.
URL https://hub.docker.com/r/library/alpine/