SecHack365

Cyrius

—Linux 非依存にコンテナをネイティブ実行する専用 OS —

#OS #Linux #コンテナランタイム #OCI #Rust

学習駆動コース 坂井ゼミ
川崎晃太朗

GitHub リポジトリはこちら→
https://github.com/n4mlz/Cyrius

● Linux はコンテナ第一の設計ではない！

Linux は元々コンテナが存在しなかった時代からあり、Linux において「コンテナ」は 後付けの機構です。コンテナは小さい機能の寄せ集めで作られており、カーネルはその機能群のみを提供します。そのため、歴史的経緯によりコンテナの分離機構は煩雑になっています。私はコンテナがとにかく好きなので、このような Linux がコンテナが第一の設計ではない状況に対しけしからん！と思いました。そこで制作したのが、この Cyrius という自作 OS です。

● Linux コンテナの動作には本当に Linux が必要？

① コンテナイメージから生成される OCI Bundle は、動作させたいコンテナの環境を定義するファイル群である。この OCI Bundle では “環境” のみを定義しているが、この “環境” さえ実現できれば前述した Linux の機能を使ってコンテナを作成することを必ずしも求めていない。

② Linux コンテナ（プロセス）は OS の機能を利用するために syscallを発行するが、この syscall の互換性さえ満たせばこのプロセスは下が Linux なのか互換性を持った別の何かなのかは気にしない。

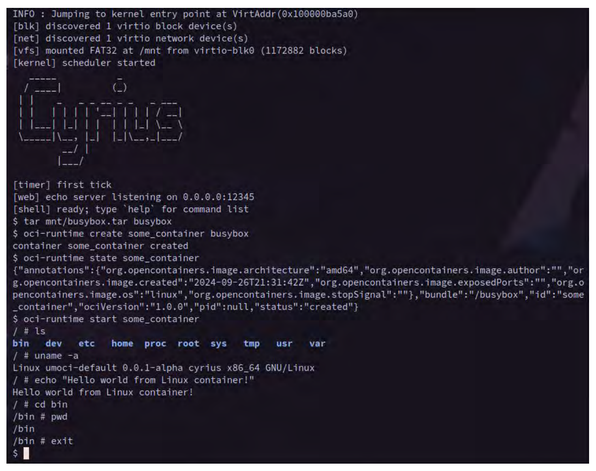
➡ということは、「カーネルが OCI Bundle を直接解釈し、よりシンプルな仕組みでコンテナを建てる専用 OS」が考えられそう？→ Cyrius の誕生

● Cyrius とは

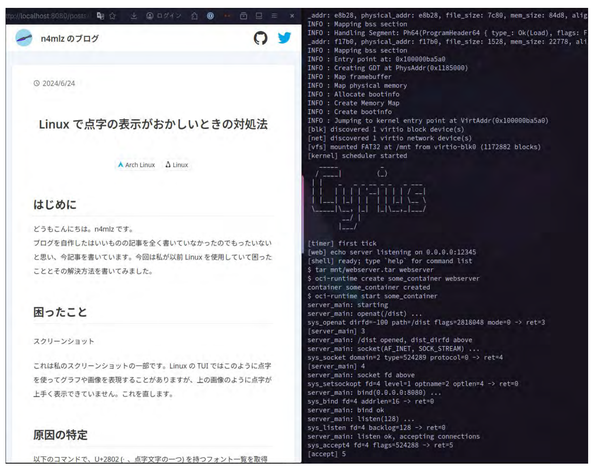
コンテナの実行に極度に特化した自作 OS であり、 それ以外の機能を持ちません。既存のコンテナの標準仕様 (OCI specification) への互換を持ち、 Docker から pull してきたイメージがほぼそのままネイティブ実行することができます。また、Cyrius ではコンテナに期待されるセキュリティと同等の 隔離性・互換性を、Linux と全く異なるカーネルの設計思想で実現します。

● Linux コンテナのネイティブ実行

以下の2つのコンテナイメージはいずれも Linux 上での実行を想定されたものですが、Cyrius 上でネイティブ実行しています。この実現のために、Cyrius は Linux バイナリ互換と OCI Runtime Specification（コンテナの標準仕様）互換を有しています。



Docker Hub から pull した busybox コンテナの実行



モダンなフレームワークを使用した Web サーバーのコンテナの実行

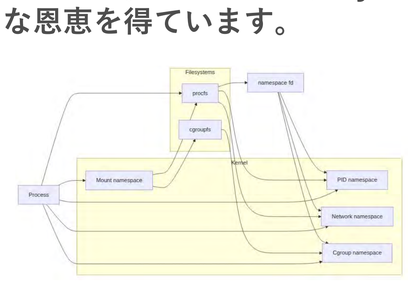
● Cyrius の特異性

Type2 Hypervisor はユーザーランド上で動作する VMM、また Type1 Hypervisor はベアメタルで OS と同様にカーネルモードで動作する VMM です。ここであえて既存のコンテナランタイム（runc, youki 等）を Type2 Container Runtime と呼ぶとすると、 Cyrius はベアメタルで OS と同様にカーネルモードで動作する Type1 Container Runtime です。このType1 Container Runtime という概念は、（私の調べる限り）Cyrius が世界初です。

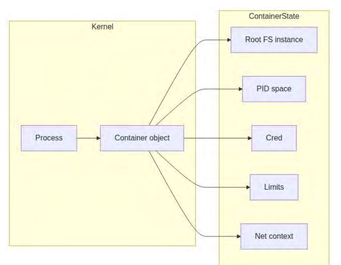
	ハイパーバイザー	コンテナランタイム
ユーザーランド (Type2)	Type2 Hypervisor 例: QEMU, VirtualBox	Type2 Container Runtime 例: runc (Docker), youki
カーネル空間 (Type1)	Type1 Hypervisor 例: VMware ESXi	Type1 Container Runtime Cyrius

● Linux vs. Cyrius

下図は、Linux と Cyrius それぞれについてコンテナを実現するコンポーネントの依存関係を表現した図です。Cyrius では環境の全てがカーネル内の Container 構造体に集約されることで、よりシンプルに「コンテナ」を表現することができています。また、Linux ではこの依存関係を構築するために syscall を最低限で20種類近く使用する必要があるのですが、Cyrius ではたった一つの syscall のみでこの環境を構築することができます。このように、Cyrius ではカーネルが直接コンテナを管理することで様々な恩恵を得ています。



LINUX



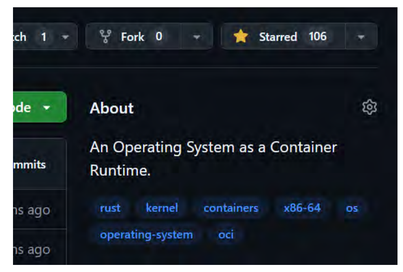
Cyrius

● Cyrius の自作 OS としての面白さ

「Linux コンテナ互換」は、自作 OS のテーマとして非常に面白いです。動かしたいアプリケーションは Linux 上でコンテナとしてビルドして自作 OS に持って行ってしまえば、あとは Linux として実行されることになります。自作 OS 用にコンパイルしたり、依存関係を頑張って揃えたりする必要はありません。また、コンテナレベルの互換の恩恵として、自作 OS の上で既存のエコシステムにタダ乗りできます。つまり自作 OS を拡充していけばいくほど自作 OS 上で生活できるようになるはずでしょう。私の夢は、これを続けていっていつか最終的に自作 OS の上で暮らすことです。

● 話題性: GitHub 3桁スター達成

自作OS×コンテナというワードが刺さったのか、Twitterで反響があり、Cyrius のリポジトリはたちまち100以上のスターを獲得しました。



● 応用可能性: ここから見える未来

○ Docker や Kubernetes とのインテグレーション
Cyrius は OCI 仕様に準拠しているので、頑張れば既存のランタイムと連携ができるはずです。containerd-shim というのを作れば実際に Docker のバックエンドとして呼び出せるようになる見込みです。

○ Cyrius で構成された Kubernetes クラスター
上記を満たすと、Cyrius をコンテナランタイムとして使用した Kubernetes クラスターが建てられることになります。自作 OS を使用した自作クラウド基盤などまだ誰もやったことがないのではないでしょうか。

● コンテナ実行に特化した新しいカーネルの設計思想

「コンテナを実行する専用のOS」という思い切りから、いかにシンプルな仕組みでコンテナの環境を実現するかという設計思想に特に強くこだわりました。

○ コンテナ用OSの新しい世界
コンテナランタイムがカーネルに統合し、リソース管理や環境の隔離に関して OS レイヤから独自の機構を備える。「コンテナはネストしない」という強い仮定を置き、コンテナランタイムが動作するホストとそれによって管理されるコンテナの2つの世界に分離されるというモデルで全てを考える。Namespace は階層構造を作らない。

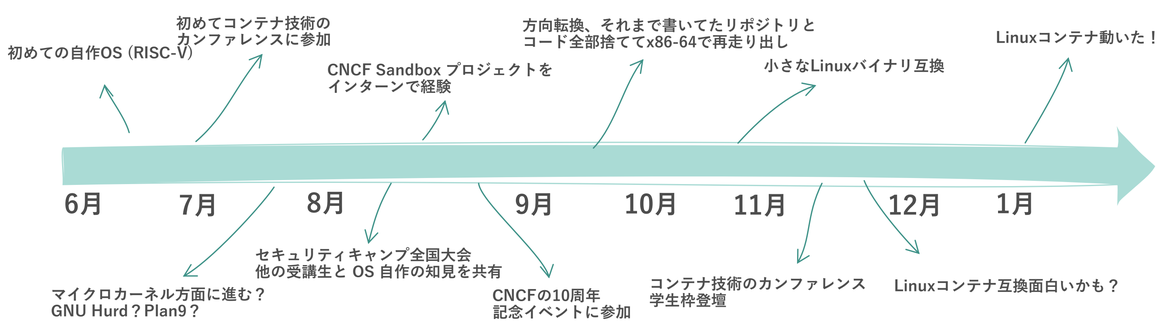
○ コンテナのカーネルオブジェクト化
コンテナはカーネルオブジェクトであり、コンテナの状態やライフサイクルは OS が直接管理する。Container は「プロセスが動作する環境の定義」であり、必ずしも所属するプロセスを必要としない。
Container 構造体には rootfs や capability、リソース制限などの環境記述情報を持たせる。プロセス空間やルートファイルシステムへのハンドルなどをフィールドに持つ。

○ ホスト/コンテナプロセスの明確な区別と2つの syscall テーブル
全てのプロセスはホストプロセスとコンテナプロセスに分類され、ホスト用の syscall テーブルと Linux バイナリ互換用の syscall テーブルを持つ。これらを Process 構造体の abiフィールドを見てsyscall テーブルを切り替える。ホストの syscall はコンテナランタイムを操作するための syscallテーブルのみに限られ、syscall は OCI ネイティブな単位である (create, start, stop 等)。そのため例えばコンテナの生成は 1 つの syscall で済み、コンテナを生成するまでに余計なカーネル空間とユーザー空間の処理の往復をしない。

○ VFS の完全な分離
コンテナ用ファイルシステムはホストファイルシステムと完全に並列なインスタンスとして生やし、プロセスは所属 Container 経由でのみ VFS にアクセスするため、chroot 的な jailbreak パターンを構造的に存在させない。また、pivot_root のような FS の挿げ替えなどの複雑な操作を要求しない。

● SecHack365 での1年間

これらの開発の全てがスムーズに進んだわけではありませんでした。SecHack365 開始時点で OS 自作未経験であったため、メインアイデアに辿り着くまでに途中で全てのコードを捨て0から書き直したり、またコンテナ方面でもOSSコントリビューションを行ったり、カンファレンスで登壇したりしていました。



この場をお借りしまして、この一年間にわたり大変お世話になりました SecHack365 事務局の皆様、トレーナーの皆様、アシスタントの皆様、そして共に学び合いましたトレーニーの皆様に、心より感謝申し上げます。

NICT National Institute of Information and Communications Technology