卒業論文　2019 年度（平成 30 年度）

卒論

慶應義塾大学 環境情報学部

石川 達敬

徳田・村井・楠本・中村・高汐・バンミーター・植原・三次・中澤・武田
合同研究プロジェクト

2020 年 1 月

卒業論文　2019 年度（平成 30 年度）

卒論

論文要旨

アブスト

キーワード

OS

慶應義塾大学 環境情報学部

石川 達敬

Abstract Of Bachelor's Thesis　Academic Year 2019

Title

Summary

abst

Keywords

OS

Bachelor of Arts in Environment and Information Studies
Keio University

Tatsunori Ishikawa

目 次

図 目 次

表 目 次

第1章　序論

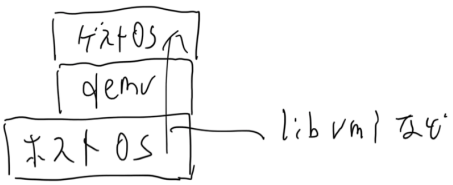
1.1　背景

コンピュータの管理者は，動作中，あるいはカーネルパニックなどによって停止したコンピュータの情報を監視・解析することが必要となる場面がある．動作中のコンピュータ自身に対しては，同一ホスト内の top コマンドや ps コマンドを用いて，プロセスの一覧を得たり，gdb コマンドを用いてプロセスをトレースし，プロセスの状態を把握する．論理的に停止したコンピュータに対しては，kdump と呼ばれる機構を通してメモリダンプを静的に解析し，原因の究明をする．また，状態を監視したいホストを物理的なマシンではなく，仮想マシンとして起動し，qemu や Xen などの基盤上で libvmi などを通して状態を解析する手法も存在する．

仮想マシン上に起動したホストに対する解析手段としては，上述の通り手段が豊富に揃っている．

これらの課題を解決する手法の一つとして，RDMA NIC を用いた解析手法が，（sora さんの論文で）発表された．

図 1.1: libvmi を用いる際のアーキテクチャ



1.2　着目する課題

Linux カーネルのバージョン，System.map の情報および config の情報を知らなければ，メモリのみから正しくコンテキストを復元していくことはできない．

libtlp の論文（もうちょっとちゃんと書く）で紹介されている process-list.c は System.map を引数として渡し，init$_t ask$ の行を $read$ することで $init_t ask$ の仮想アドレスを得ている．また，カ

ーネルコンフィグに依存するマクロの値や，使用する関数などもハードコーディングされており，論文の環境以外で動かすことが容易ではない．

1.3 目的

そこで本研究の目的として，問題の章であげた 3 つの情報，Linux カーネルのバージョン，System.map の情報およびカーネルコンフィグのうち，System.map およびカーネルコンフィグの情報を RDMA NIC を用いて復元する．また，Linux カーネルのバージョンを知ることさえできれば，どのようなカーネルコンフィグを持つコンピュータに対しても，RDMA NIC を通してプロセスリストの一覧を取得できることを実証する．

この目的を達成するため，本研究では，いくつかの段階に分けてネットワーク越しにあるコンピュータに対して監視･解析を行っていく．第一の工程として，RDMA NIC を用いて，監視対象ホストのメモリを全探索し，メモリに落ちている System.map のうち，init$_t ask$ が配置されている仮想アドレス空間に関する情報と，$Linux$ カーネルにおける$_{p}hys_{a}ddr$ 関数，$task_{s}truct$ 型を決定するためのカーネルコンフィグに関する情報を収集

次の工程として，収集したカーネルコンフィグを元に手元のコンピュータで Linux カーネルのソースコードに対してプリプロセスの処理を行い，task$_s truct$ 型を確定する．さらに，ソースコード上にある$_{p}hys_{a}ddr$ の実体を収集する．

最後に，この工程で得られた情報をもとに，libtlp で提唱されている手法を用いて，プロセスの一覧を正しく取得できることを確認する．

1.4 本論文の構成

こんなことは最後に書く．！！！！！！！！！！！！

2 章では、本研究で使用する libtlp と，その基盤として使用している RDMA について述べる．

??章では、RDMA を通してネットワーク越しにあるコンピュータを監視・解析を行うための実行環境の構成に関して述べる．

4 章では、本研究で実装したものについて述べる（加筆）

5 章では、本研究における評価として，Linux カーネルのバージョンのみが与えられた状態でプロセスリストの一覧を取得できることを示す．

6 章では、本研究に関する結論と，今後の課題について述べる．

第2章　関連技術

本章では，本研究で基盤技術として使用する RDMA（Remote Direct Memory Address），および libtlp に関して述べる．

2.1　vm を用いた解析

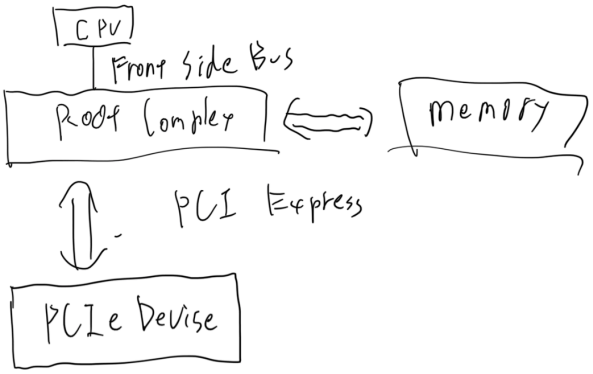
2.1.1　qemu

2.1.2　libvmi

2.2　RDMA

ここに RDMA の説明をかく

図 2.1: PCI Express



2.2.1　Infiniband における RDMA 実装

2.3　libtlp

このセクションは実装の章に移動

ここに libtlp の説明，とりわけ process-list.c で行われている処理に関して書く．

libtlp 本来の目的は，PCIe デバイスの開発プラットフォームであるが，その機能の一つとして，DMA message と ethernet パケットを相互変換する機能があり，それを利用して，RDMA を行っていく，みたいなことをかく

第3章　アプローチ

1.3 で，本研究の目的を，動作中のコンピュータのメモリのダンプをリアルタイムで解析することで，コンピュータの状態をリモートホストから知ることができるようにする，と定義した．

そこで本章では，メモリのダンプをリアルタイムで取得．解析する上で前提となる情報と，この手法における課題について述べる．

3.1　オペレーティングシステムのコンテキスト

コンピュータの状態，すなわちオペレーティングシステムの動作中におけるホストの状態は，コンピュータ内部におけるレジスタの値および，内部から参照できる仮想アドレス空間上に保持されている．その例を下に示す．

あるプロセスを実行する際に，プロセッサはインストラクションポインタレジスタの命令を読み込み，逐次実行をしていく．call 命令などで別の関数を呼ぶ際には，その時点におけるインストラクションポインタレジスタの値をメモリ上に退避し，関数が終わった際に，呼び出し元に返るように設定されている．実行コードが整合性を保っているかは，実行可能ファイルを生成したコンパイラの責務なので，本論文では述べないが，プロセッサはプログラムの実行を行う際，レジスタの値を参照，退避，復帰，上書きさせることで，状態を保持，進行させていると言える．

プロセッサがレジスタの値を参照しそれを仕組みは，カーネルのコードを実行する際にも言える．ここでは，オペレーティングシステムから見たコンピュータの状態として，プロセスの切り替え処理，コンテキストスイッチにおける処理の流れを述べる．コンテキストスイッチとは，割り込み処理などによって定期的に呼ばれるプロセススケジューラーから呼ばれる機構である．この機構は，実行中のプロセスの状態，すなわち，各レジスタの値および仮想アドレス空間に関する情報などをカーネルが管理しているメモリ上にあるデータ構造の中に退避する．

本セクションのまとめとして，コンピュータの状態は，ある瞬間においてはレジスタの値であり，この状態を保存する際は，メモリ上にレジスタの値を退避させていることを述べた．

3.1.1　task$_s$*truct* 構造体

3.1 でコンテキストスイッチにおいて，退避されるプロセスの情報は，対応したデータ構造に退避されると述べたが，この時に使用されるデータ構造が task$_s$*truct* 構造体である．*task*$_s$*truct* 構造体には，一つのプロセスの情報の全て（？）が格納されている．その中には，仮想アドレス空間に関する情報を保持する *mm*$_s$*truct* 構造体を参照するフィールドも存在する．

コンテキストスイッチ時には，task$_s$*truct* 構造体に保持されている情報をレジスタに復帰させることで，中断される直前の情報を復元している．

3.1.2　オペレーティングシステムのビルドにおけるコンフィグ

task$_s$*truct* 構造体をはじめとして，*Linux* カーネルの変数や型，関数は，様々なアーキテクチャやカーネルコンフィグに対応するため，マクロによって分岐されている．この分岐が確定するのは，*Linux* カーネルをビルドするときであり，構造体のメンバへのアクセス，関数のアドレスなどはコンパイラが保証している．

実際のカーネルのバイナリは，vmlinux としてコンパイルされた後，strip され bzImage となる．ユーザーが作成したカーネルモジュールなどで関数を呼び出す際は，シンボルとアドレスの変換表である‘/boot/System.map‘を参照し，仮想アドレスを得たのち，実際にメモリにアクセスする際に物理メモリアドレスを算出しアクセスしている．

3.1.3　ホスト自身によるレジスタやシンボルの参照

3.1 で述べたように，オペレーティングシステムでは，レジスタの値などを退避する際，そのプロセッサ自身が’push’ 命令などを用いてメモリにアクセスできる

レジスタを参照して，かつ mmu すら自分で参照ができる．

例として，push 命令が実行された際の流れを述べる．

CPU レジスタの現在の値は直接知ることができないため，例えばプロセスの一覧を取得したい場合は，コンテキストスイッチ時に退避された値を辿っていく必要がある．しかし，上述の通り task$_s$*truct* はビルドされた際のカーネルコンフィグによって，どのメンバが先頭アドレスからどのオフセットに保持されているかは変動する．

第4章　実装

実装がまだ完成してないので，空想です．

4.1　実験環境

本研究で実装を行う環境は，RDMA NIC が刺さった監視対象ホストと，本研究における実装を実行するホストの 2 台で構成する．

監視対象ホストは，Linux 4.15.0-72-generic の ubuntu であり，PCIe デバイスとして，FPGA ボードが刺さっている．この FPGA ボードは，2 章にて述べたように，dma message と ethernet パケットを相互変換する機能を有しており，インターフェースとして，dma$_read$ 関数と dma_write 関数が $libtlp$ に用意されている．また，この $FPGA$ ボードには，IP アドレスとして，192.168.10.1 を静的に振ってある．

実装を実行するホストは，Linux 4.19.0-6-amd64 の Debian buster であり，光ファイバーケーブル (名称はあとで修正) が刺さる NIC を刺している．この NIC には IP アドレスとして，192.168.10.3 を静的に振ってある．監視対象ホストに対して RDMA を実行する際は，dma$_read$ 関数，あるいは dma_wirte 関数を通して 192.168.10.1 に対して IP パケットを送信している．

図 4.1 にて，全体図を書く

4.2　物理アドレスを指定したメモリの値の取得

ここにどのように叩くと値が返ってくるかを書く

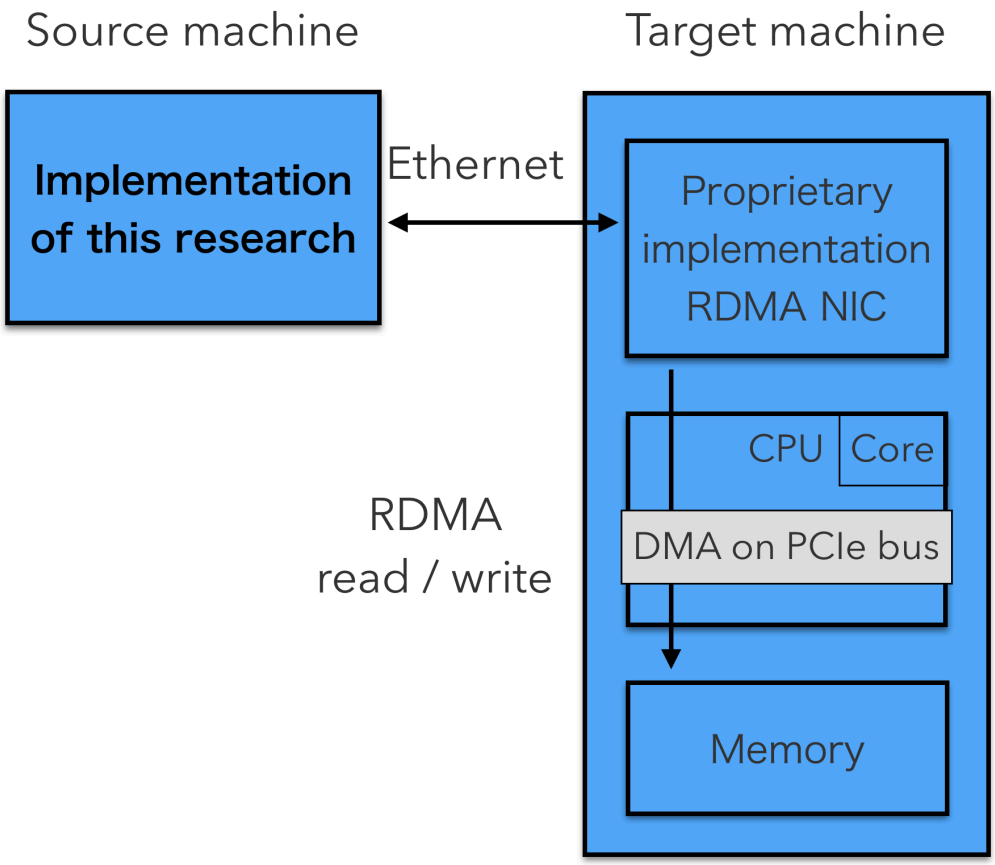
4.3　実装の全体

ダンプしてくるということを書く．物理アドレスのマッピングに関しても

次の工程として，収集したカーネルコンフィグを元に手元のコンピュータで Linux カーネルのソースコードに対してプリプロセスの処理を行い，task$_struct$ 型を確定する．さらに，ソースコード上にある $_{phys_addr}$ の実体を収集する．

最後に，この工程で得られた情報をもとに，libtlp で提唱されている手法を用いて，プロセスの一覧を正しく取得できることを確認する．

図 4.1: 全体


Source machine
Target machine
Implementation of this research
Ethernet
Proprietary implementation RDMA NIC
CPU
Core
DMA on PCIe bus
RDMA read / write
Memory

4.4 工程 1

第一の工程として, RDMA NIC を用いて, 監視対象ホストのメモリを全探索し, メモリに落ちている System.map のうち, init$_{t}ask$ が配置されている仮想アドレス空間に関する情報と, $Linux$ カーネルにおける$_{p}hys_{a}ddr$ 関数, $task_{s}truct$ 型を決定するためのカーネルコンフィグに関する情報を収集することは 2 章で述べた.

（本セクションでは，その実装を詳しく書く．ダンプしまくるやつの説明をここに書く）

4.5 工程 2

収集したカーネルコンフィグを元に手元のコンピュータで Linux カーネルのソースコードに対してプリプロセスの処理を行い，task$_{s}truct$ 型を確定する．さらに，ソースコード上にある$_{p}hys_{a}ddr$ の実体を収集する部分に関する実装をより詳しく書く.

4.6 工程 3

最後に，集められたデータをもとに，process-list を改造したものに関する説明をここに書く

第5章　評価

評価をしたい

5.1　評価手法

カーネルのバージョンのみわかる状態から，正しく ps aux と同じような出力を得られるかどうかを評価とする．

5.2　評価

未評価

第6章　結論

アブスト

6.1　本研究の結論

結論

6.2　今後の課題

課題はたくさんある

謝辞

アドバイスをくれた全員に感謝