DMA 機構の故障に対するデバイスドライバの耐性検証

畑 中 俊 輝[†]　味曽野 雅史[†]　品 川 高 廣[†]

1. はじめに

デバイス故障は長期にシステムを運用する上で避けることのできない問題である．デバイス故障が生じた場合，デバイスドライバは適切なエラーハンドリングを実施し，OS の処理を継続できるようにすることが求められる．しかし，多くのデバイスドライバはデバイス故障を想定しておらず，デバイス故障に対する耐性が低いことが知られている[1),2)]．

デバイス故障に対する耐性が低いデバイスドライバを見つける方法として，デバイスとデバイスドライバとの間の I/O アクセスに対して Fault Injection による擬似的な障害を発生させることで不具合が発生するデバイスドライバを検知する手法が提案されている[2),3)]．しかし，最近のデバイスではディスクリプタと呼ばれるメモリ上の構造体を介してデバイスドライバとやり取りを実施するものがある．この場合デバイスは DMA(Direct Memory Access) により，CPU の I/O 命令を介さずにディスクリプタのデータの読み書きをおこなう．先行研究ではこのようなディスクリプタ上のデータの破損に対するデバイスドライバの耐性を検査することができないという問題がある．

本研究ではデバイスとデバイスドライバの間のやり取りで用いられるメモリ上のディスクリプタに対して Fault Injection をおこなう手法を提案する．これによりデバイスの DMA 機構の故障によりディスクリプタが破損した場合に，正しく処理を実施せずに OS をクラッシュさせてしまうデバイスドライバの問題を発見することができる．本研究により OS やハイパーバイザのデバイスドライバの信頼性をより向上させることが可能となる．

2. 提案手法

図 1 に提案手法の構成を示す．提案手法では，まず検査対象となるデバイスが利用するディスクリプタ箇所を特定する．その後，その特定した領域に対して Fault Injection を実施し，デバイスドライバが DMA 機構の障害に対してどの程度の耐性を持っているかを検証する．

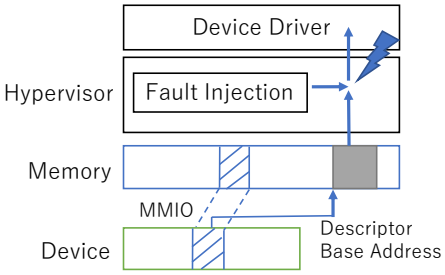

図 1　提案手法の構成

以下でそれぞれのステップの詳細を述べる．

2.1 ディスクリプタ領域の特定

デバイスが利用するディスクリプタ領域を特定する方法は大きく二つ考えられる．一つはデバイスの仕様を参照することである．ほとんどの場合，デバイスが利用するディスクリプタのアドレスは，デバイスの特定の MMIO レジスタに格納される．Fault Injection 開始時にデバイスが初期化されている場合，単純にこのレジスタ値を参照することでディスクリプタ領域のアドレスが得られる．Fault Injection 開始時にデバイスが初期化されていない場合（例えば，Fault Injection 中のワークロードとしてデバイスドライバのロードを選択した場合），ハイパーバイザのネステッドページングの機能を用いてデバイスドライバの MMIO レジスタアクセスを補足し，ディスクリプタのアドレスの値を得る．ワークロード実行中のディスクリプタ領域の変化の検知にも同様の手法を用いる．

また，デバイスの挙動を観察することでデバイスが利用するメモリ領域を取得し，ディスクリプタ領域を特定する手法もある[4)]．OS によってはデバイスドライバが DMA 用に利用するメモリを確保するための特別な関数を用意している場合がある（例：Linux における `dma_alloc_coherent()`）．検査時に OS と協調できる場合，このような情報を利用することで効率よくディスクリプタ領域を特定することができる．この方法はデバイスの仕様情報を必要としないという利点がある．ただし，この方法では偽陽性・偽陰性が存在する可能性があることに注意が必要である．

2.2 Fault Injection による検査

Fault Injection は FaultVisor[2)] の手法を応用する．

† 東京大学
The University of Tokyo

この手法では Fault Injection 対象の OS をハイパーバイザ（FaultVisor）上で動作させる．そして，ネステッドページングの設定により OS のデバイスドライバがデバイスの MMIO 領域にアクセスした際，意図的に例外を発生させハイパーバイザ側へ制御を移す．

ハイパーバイザはデバイスドライバのアクセスが read であった場合，本来とは異なる値をデバイスドライバへ返すことで Fault Injection をおこなう．この際軽量な準パススルーハイパーバイザ[5] を利用することで，実環境に近い環境でデバイスドライバの検査をすることができる．今回の場合，Fault Injection の対象はデバイスの MMIO 領域（図 1 の斜線の領域）ではなく，前述の手法で特定したディスクリプタが存在するアドレスの領域（図 1 の灰色の領域）である．

実際の検査では特定のワークロードを実行した上で Fualt Injection をおこない，デバイスドライバの挙動を観察する．この際にカーネルパニックやハングアップなどが観測された場合，デバイスドライバのエラーハンドリングが不適切であったと判断する．

3. 実　　装

現在 NVMe(Non-Volatile Memory express) デバイスを検査対象として，FaultVisor[2] を拡張する形で提案手法の実装を進めている．以下で NVMe デバイスの基礎及び，現在の実装状況について説明する．

3.1 NVMe デバイス

NVMe[6] は PCIe 接続のストレージメディア接続規格である．今日次世代のストレージとして NVMe SSD が普及しつつある．

NVMe ではデバイスドライバとデバイスとの通信のために Admin Queue 及び Command Queue の二種類のキューを利用する．前者が管理用，後者がデータ通信用である．さらに，それぞれのキューは Submission Queue(SQ) 及び Completion Queue(CQ) に分けられる．SQ がデバイスドライバからデバイスに対するメッセージを格納するキュー，CQ その逆方向のメッセージを格納するキューである．いずれのキューもメモリ上に配置され，デバイスとは DMA を介してやりとりされる．キューが格納するディスクリプタのフォーマットは仕様により定められており，CQ の場合はデバイスからのステータスを表すフィールドの他，対応する SQ へのポインタなどが含まれる．

3.2 ディスクリプタ領域の特定

Fault Injection 対象の対象とするのは，デバイスからのデータを格納する Completion Queue 上のディスクリプタ領域である．NVMe の場合 MMIO レジスタに各キューのベースアドレスが格納される．今回はこのレジスタの情報を参照することで Fault Injection 対象となるディスクリプタ領域を求める．例えば，NVMe の MMIO レジスタの 30h 番地に Admin Completion Queue のベースアドレスが格納されている．

3.3 Fault Injection による検査

今回ワークロードとしてはデータの読み書きのベンチマーク及び，デバイスドライバのロード及びアンロードを検討している．Fault Injection の値の改変手法としては常に固定値を返したり，乱数やビット反転の結果を返したりすることで DMA 障害をエミュレートする．本手法でデバイスドライバの異常を検知した場合，改変した CQ のフィールド情報を実際のソースコードの問題箇所の特定に役立てることができる．

4. 関 連 研 究

デバイス故障に対するデバイスドライバの耐性を高める研究として，Carburizer[1] は静的解析により，デバイスドライバの不適切なエラーハンドリング箇所を自動修正するシステムを提案している．また，ハイパーバイザを用いたデバイスドライバへの Fault Injection を実施することで OS やハイパーバイザのデバイスドライバのデバイス故障に対する耐性を検査する研究がある[2),3]．しかし，これらの研究ではデバイスドライバとデバイス間の I/O のみを対象としており，DMA 領域に対する検査は対象外である．本手法は FaultVisor[2] を拡張する形でこれらの研究の対象外であった DMA 領域に対して Fault Injection を実施し，DMA 機構の故障に対するデバイスドライバの耐性の検査をおこなう．

5. まとめと今後の予定

本稿では DMA 機構の故障に対するデバイスドライバの耐性を検証するためのシステムを提案した．現在 NVMe デバイスを対象に提案手法の実装を進めている．

参 考 文 献

1) A. Kadav, et al. Tolerating Hardware Device Failures in Software. SOSP'09.
2) S. Takekoshi, et al. Testing Device Drivers Against Hardware Failures in Real Environments. SAC'16.
3) M. Misono, et al. FaultVisor2: Testing Hypervisor Device Drivers against Real Hardware Failures. CloudCom'18 (To Appear).
4) D. Cotroneo, et al. MoIO: Run-time Monitoring for I/O Protocol Violations in Storage Device Drivers. ISSRE'15.
5) T.Shinagawa, et al. BitVisor: A Thin Hypervisor for Enforcing I/O Device Security. VEE'09.
6) NVM Express, Inc. Specifications - NVM Expres. `https://nvmexpress.org/resources/specifications/` (Viewed on 2018-11-18).