全体ミーティング
2009/4/7

D3　西川 賀樹

本日紹介する論文

- Fast Secure Virtualization for the ARM Platform
  - Daniel R. Ferstay
  - University of British Columbia
  - 2006
- ARMにおけるVMM実装についての論文
  - 前回ARM上でのVMM実装の論文を2つ紹介
  - おそらく主な論文はこの3つのみ

概要

- PDAやスマートフォン等のモバイル機器の性能が大幅に向上、またネットワークにも接続
  - 動作するアプリケーションも豊富となり、汎用OSが用いられるようになっている
- その結果セキュリティリスクも増大
- VMMはそのようなモバイル機器において有効
  - セキュリティの強化、リソースの管理、OS等の開発・デバッグ

目的

- ARM（StrongARM:SA-110）上で動作するXenを実装
  - ARMは組み込みにおいて最も普及している、またパフォーマンスと低消費電力を両立
- VMMでのセキュリティ確保は有効
  - ユーザレベル・カーネルレベルどちらで動作するアンチウイルスソフトにしても、カーネルを乗っ取られると対処できない
  - VMMによるセキュリティ確保ついては色々議論はあると思いますが

Intel x86 vs. StrongARM

- x86とStrongARMの特徴から、Xenのparavirtualized interfaceはどのようになるか
  - Support for a secure VMM
  - CPU
  - Memory Management
  - Device I/O

Support for a secure VMM

- x86
  - 4つの特権レベルを持ち、リング機構による保護
  - センシティブ命令が存在
    - sensitive register instructions（PUSHF、POPF等）
    - protection system references（PUSH、POP等）
- StrongARM
  - 2つの特権レベルを持つ
  - センシティブ命令は存在しない
    - 以前紹介した論文にはセンシティブ命令が存在するとありましたが
- StrongARMの方が実装は容易

CPU（x86）

- Protection
  - ゲストOSを修正し、特権・センシティブ命令はハイパーコールによってXenが行う
- Exceptions
  - 例外ハンドラはほとんど元々のゲストOSと同じだが、ページフォルトに関しては修正したものを使用
- System Calls
  - ソフトウェア例外、高速化のためXenを経由せず直接実行される
- Interrupts
  - 割り込みはXenからゲストOSに非同期に通知されるイベントに置き換えられる
- Time
  - サイクルカウント単位の粒度

CPU（StrongARM）

- Protection
  - ゲストOSとアプリケーションが同じユーザモードで動作するので、仮想的な特権レベルを追加しゲストOSを保護
- Exceptions
  - x86と同様だがページフォルトに関しても修正の必要はない
- System Calls
  - x86（32ビット）とは違いページ単位の保護を行うので、ハイパーバイザ経由で実行する必要がある
- Interrupts
  - x86と同様のイベント通知システム
- Time
  - いくつかのStrongARMではサイクルカウンタは提供されていない

Memory Management・Device I/O（x86）

- Memory Management
  - コンテキストスイッチの度に全てのTLBをフラッシュする必要がある
  - Xenへの遷移の際にTLBフラッシュを回避するため、アドレス空間の高位64MBをXenが使用
  - リング機構によるセグメントの保護
- Device I/O
  - イベント通知システムを用い、データは非同期バッファリングの共有メモリにて転送される

Memory Management（ARM）

- x86との違いはページレベルの保護を行う
- x86と同様に高位64MBはXenが使用し、スーパーバイザモードでのみ読み書き可能にする
- ゲストOSはゲストOSとアプリケーションのメモリ領域を読み書き可能でマップする
- アプリケーションはゲストOSのメモリ領域をマップしない
- ゲストOSとアプリケーションとの遷移において、Xenによるページテーブルの切り替えが必要

ARM Domains（1）

- ARMではドメイン（Xenのドメインとは関係ない）によるアクセス制御が可能
- 16のドメインを定義可能
- 各ページテーブル・TLBエントリにはアクセス可能なドメインを設定可能
- マネージャ、クライアント、アクセス不可
- domain access control register（DACR）に設定

ARM Domains（2）

- アドレス空間がオーバーラップしない限りはTLB・キャッシュのフラッシュが必要なくなる
  - オーバーラップした場合、異なるドメインIDのアクセス不可であるページにアクセスするため例外が発生
  - その時にはTLB・キャッシュをフラッシュ
- ページテーブルの切り替えなしに、ゲストOS・アプリケーション間でのコンテキストスイッチが可能
- （ARMv6からはASIDが使用可能）

StrongARM PID Relocation・Device I/O

- StrongARM PID Relocation
  - FASS（Fast Address Space Switching）によりアドレス空間の衝突を回避が可能
  - ただし定義可能なアドレス空間は64まであり、各アドレス空間は32MB
  - 32MB制限では厳しいが、パフォーマンスを考え将来的に導入予定
  - （ARMv6におけるFCSE（Fast Context Switch Extension）、定義可能なアドレス空間は128、ASIDが導入されたため使用は推奨されていない）
- Device I/O
  - x86と同じ

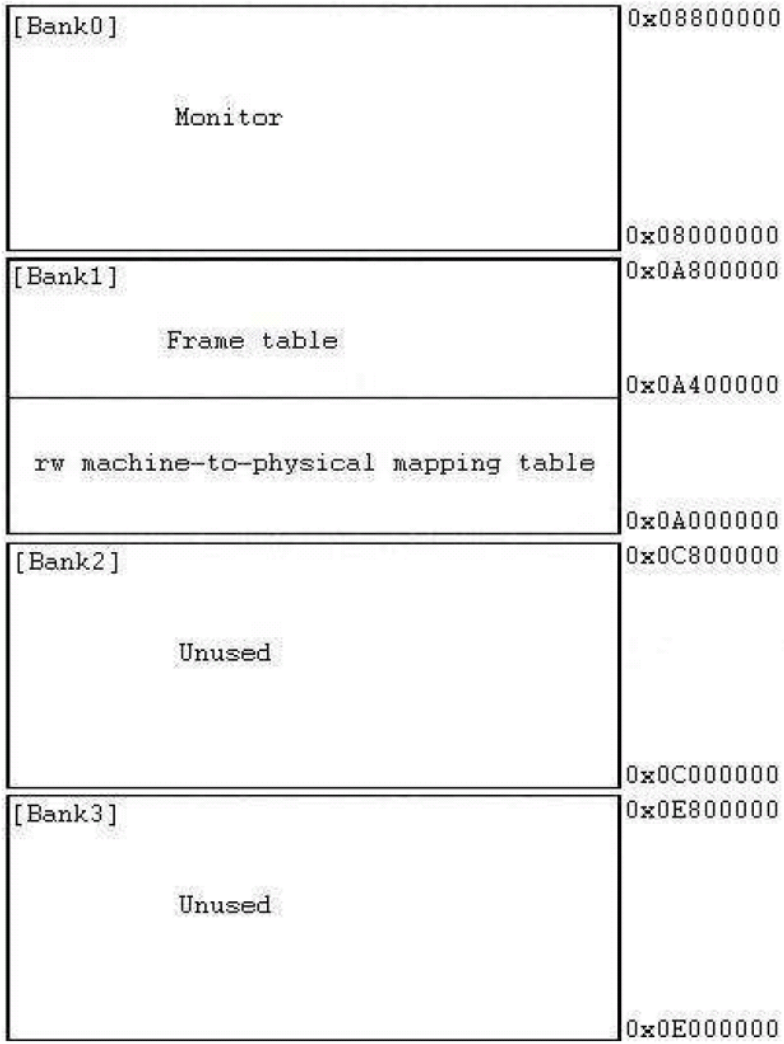
Design and Implementation

- Xen 1.2をStrongARMに移植
- プラットフォームはDNARD（Shark）
  - CPU：SA-110 233MHz　メモリ：SDRAM 32MB
- Xen 1.2を用いた理由はGCCのバージョンによる制限

Hypervisor boot code

- ハイパーバイザのロードはLinuxカーネルと同様の手法で行う
- ブートの流れ
  - イメージファイル（ゲストOSも含まれている）をTFTPにより取得
  - ファームウェアによりイメージファイルをロードし、ブートローダにジャンプ
  - ハイパーバイザを展開し、先頭部分にジャンプ
  - ハイパーバイザのブートコードにより各種初期化処理を行う

Hypervisor direct mapped region



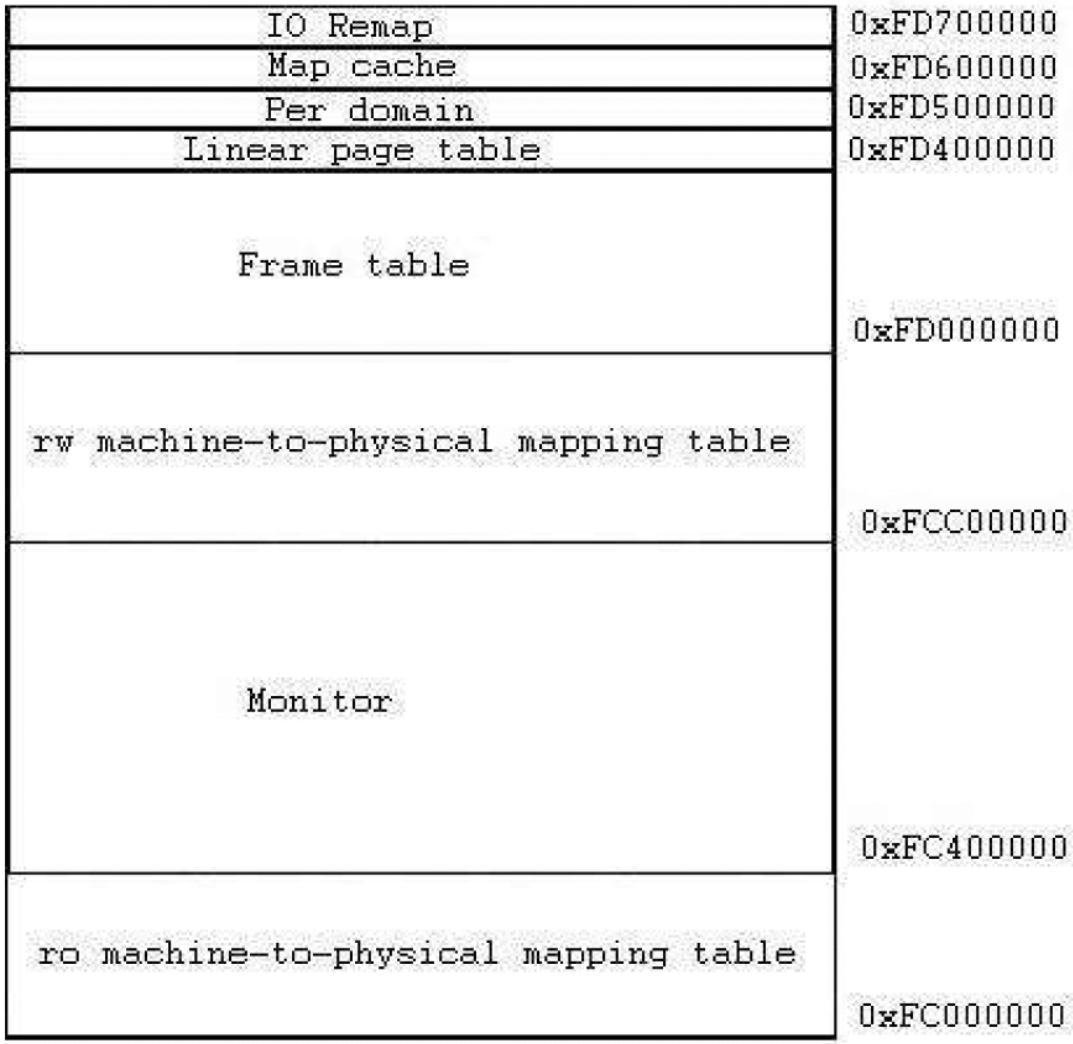
Idle task

- ダイレクトマップのメモリ領域の初期化が終わるとIdle taskの下でデータ構造の初期化
- 各ドメインはタスクとして管理され、それぞれページテーブルとスタックを持つ
- 実行可能なドメインがない場合Idle taskがロードされる

Frame table

- VMに割り当て・使用されるページの管理に使用
- Frame tableはリンクリスト（pfn_info）であり、各ノードは対応する物理ページとそのページを使用するドメインの情報を保持
- Xen/x86では連続したメモリ領域を想定
  - DNARDは不連続なので、メモリバンク2を使用するようにハードコード
  - phys_addr = frame_num * page_size + phys_offset

Hypervisor address space layout



Trap handling・Real time clock

- Trap handling
  - 未定義命令、ソフトウェア割り込み、プリフェッチアボート、データアボート、IRQ、FIQ（高速割り込み）、アドレス例外
  - CPSRとPCからハイパーバイザ・ドメインどちらによるものか判断し、適切に処理
  - FIQ、アドレス例外は使用しない
- Real time clock
  - ARM LinuxのRTCドライバのコードを使用し、経過時間を測定

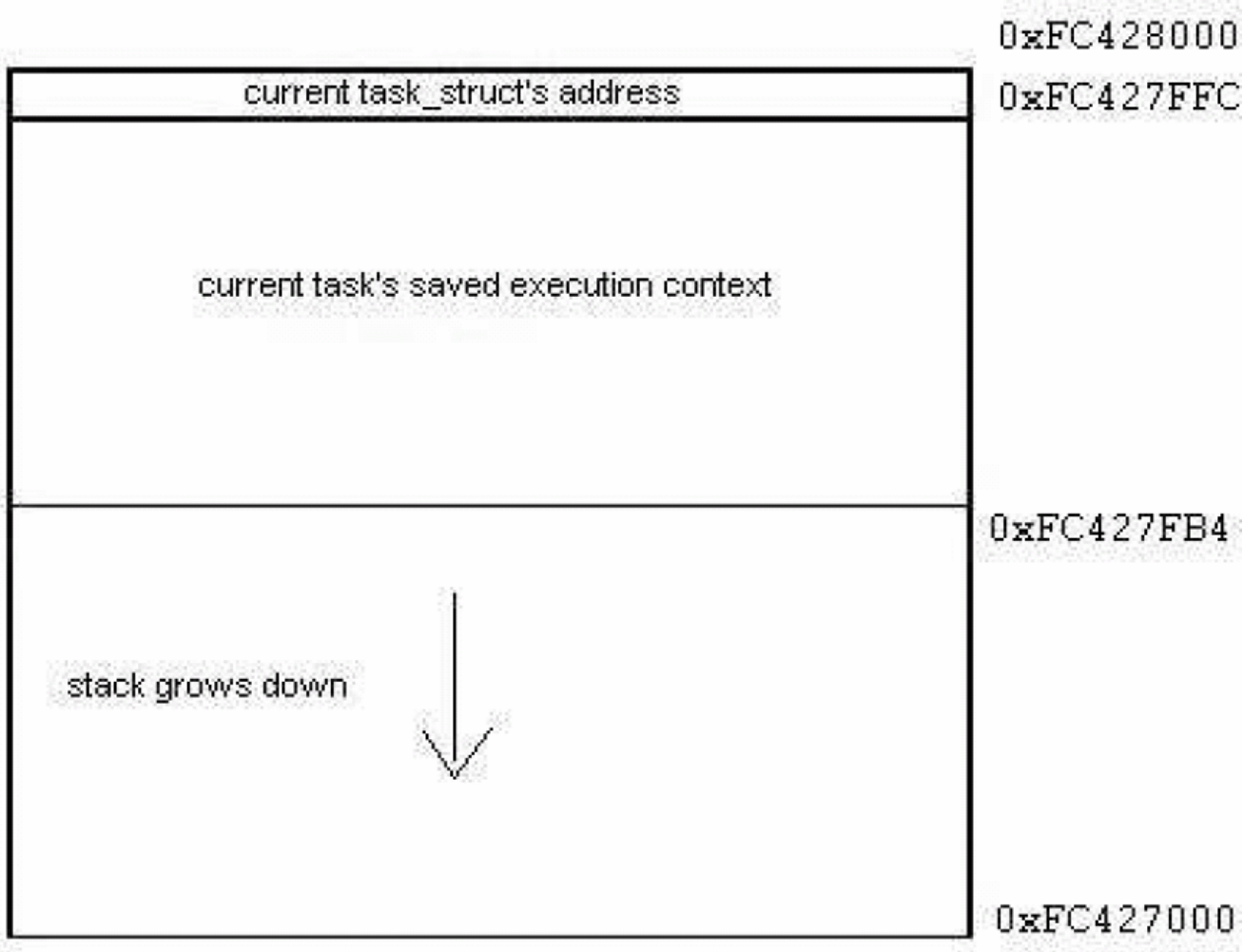
Scheduler

- RTCハードウェア割り込みハンドラを利用してソフトウェア割り込みを発生させる
  - DNARDではAPICタイマがない
- softirqのフラグをセットする
  - ARMではx86のBTSL命令のようなものはない
  - アトミックにフラグの保存、セットを行う必要がある
  - シンプルなCコードで実装

Task Management code

- ドメインはタスクとして表現
- 各状態はハイパーバイザスタックに保存
  - スケジューリング時のコンテキストスイッチやハイパーコール・イベント通知時のエントリコードにおいて使用
- Task Management codeはアセンブリで記述されている
  - パフォーマンスに大きく影響するため

Memory layout of the hypervisor's stack



Domain Execution Context Definition

```
typedef struct {
  unsigned long r0; /*working regs*/
  unsigned long r1;
  unsigned long r2;
  unsigned long r3;
  unsigned long r4;
  unsigned long r5;
  unsigned long r6;
  unsigned long r7;
  unsigned long r8;
  unsigned long r9;
  unsigned long r10;
  unsigned long fp; /*r11, frame ptr*/
  unsigned long ip; /*r12, intra-proc call scratch*/
  unsigned long sp; /*r13, stack ptr*/
  unsigned long lr; /*r14, link reg */
  unsigned long pc; /*r15, prog cntr*/
  unsigned long psr;/*status reg (mode bits/flags)*/
  unsigned long _unused;
} execution_context_t;
```

Context switch code

```
void switch_to( task_struct *prev, task_struct *next ) {

  /* get current tasks context */
  execution_context_t *ec = get_execution_context();

  /* save prev state to task_struct  */
  memcpy( prev->ec, ec, sizeof(*ec) );

  /* restore next state to hyp stack */
  memcpy( ec, next->ec, sizeof(*ec) );

  /* flush I cache and TLB; clean D cache;
   * switch page tables
   */
  write_pgdbase( next->mm.pagetable );

  /* hyp stack top contains ref to cur task */
  set_current( next );

}
```

Code for building domain0

```
task_struct* do_createdomain( int dom_id ) {
        allocate a task structure and fill in its domain id;
        allocate a page for the shared_info portion of the task;
        mark the shared_info page as shared between the hypervisor and
              the domain in the frame table;
        allocate a page for the domain's per domain page tables;
        add the task_struct to the scheduler's list of runnables;
        initialize the domains list of allocated physical pages to 0;
        map dom_id to the task_struct ptr in the task_hash map;
        return the task_struct ptr;
}
```

Code for loading domain0

- ほぼx86と同様だが、以下の点で異なる
  - 仮想ネットワーク・仮想ブロックデバイスをサポートしていないため、その作成コードがない
  - ページテーブルの構造の違い
  - 検証のためにマップキャッシュにゲストOSイメージをマップする必要がない
    - 既にハイパーバイザのイメージに含まれているので
- 複数ドメイン、またゲストOS上でのアプリケーションの実行をサポートしていない

Hypervisor entry code

- ハイパーバイザのスケジューラかゲストOSによるハイパーコールによりエントリコードが実行される
  - ハイパーコールはソフトウェア割り込み（SWI命令）を通して行われる
- イベントの通知は以下の流れで行われ、ハンドラ実行後にハイパーバイザにスイッチしない
  - 割り込み時のSP・PC・ワーキングレジスタ等ゲストOSの状態のクローン（bounce frame）を作成
  - そのクローンフレームを指すようSPを修正
  - PCをイベントハンドラを指すよう修正
  - ゲストOSが実行を再開し、イベントハンドラが実行
  - ハンドラは実行後クローンフレームをPOPし、ゲストの割り込み時の状態にダイレクトにジャンプ

実装済み機能

- hypervisor boot on the DNARD
- output via the serial console
- real time clock hardware initialization
- software interrupts and accurate timers
- notion of virtual and wallclock time
- hypervisor events used to asynchronously pass messages to the hypervisor
- guest events used to virtualize interrupts.
- hosting a single guest OS
- isolation of the hypervisor from the hosted guest OS using page level protection in conjunction with the two physical execution modes on the ARM cpu

未実装の機能

- frame table management of discontiguous physical memory
- hypervisor heap allocator management of discontiguous physical memory
- isolation of guest OSes from their hosted applications using page level protection in conjunction with the two physical execution modes on the ARM cpu plus a virtual execution mode in the hypervisor
- hosting more than one guest OS
- ethernet hardware initialization
- virtual block devices
- virtual network interfaces
- transfering data between the hypervisor and guest OSes using Xen's asynchronous
- I/O rings

評価

- 実験環境
  - DNARD Shark
    - StrongARM SA-110 233MHz　　メモリ 32MB
  - ゲストOSとしてARM用に移植したMini-OSを使用
    - domain0として動作させる
- Hypercalls、Batched Hypercalls、Delivering Events、Domain Loadingの4つの操作とメモリ消費量を測定
  - ただしスケジューラは動作させないようにハイパーバイザを修正
  - RTC割り込みハンドラはシステム時間とshared_info構造体の更新にのみ使用され、スケジューリングを起動させるsoftirqは発生させない
  - RTC割り込みの周期による制限で、7msの粒度による測定しかできないため、10^6回の操作の平均時間による測定

評価する4つの操作

- Hypercalls
  - no-opのハイパーコールを実行し測定
- Batched Hypercalls
  - HYPERVISOR_multicallによりno-opのハイパーコールをバッチ処理
  - 実際には物理メモリ量の制限からHYPERVISOR_multicallを修正し評価
- Delivering Events
  - ハイパーバイザがタイマイベントを通知し、Mini-OSがその処理を行うまでの時間を計測
  - ただしMini-OSのタイマハンドラはreturnを行うだけ
- Domain Loading
  - domain0の作成、ロードの時間を測定

4つの操作の実験結果

Operation	Time per 10^6 operations (ms)	Average Time per operation (μs)
Hypercall	504	0.504
Batched Hypercall	98	0.098
Event Delivery	1960	1.960
Domain Loading	n/a	49.0

x86とARMにおけるメモリ消費量の比較

Region (MB)	ARM	x86
Monitor	8	16
Machine-to-physical Mapping Table	4	4
Frame Table	4	20
Total Memory Consumed	16	40