Technology Consulting Company
Research, Development &
Global Standard

The Current Status of BitVisor 1.5(仮)

榮樂 英樹
株式会社イーゲル

2014-11-21 BitVisor Summit 3

BitVisor 1.5(仮)で新しくなること

igel

- PCIデバイスドライバー関連
- TCP/IPスタック
- 性能改善
- デバッグ用新機能
- その他バグ修正など

PCIデバイスドライバー関連

PCIデバイスドライバー関連目次

igel

- 従来のプログラムと設定方法
- これまでの問題点と要望
- 解決策
- デバイスの指定方法
- プログラムの比較
  - 新しい部分の説明
- Configurationの比較
  - 記述方法の説明
- デバイス一覧・ドライバー割り当て確認機能
- その他のPCIドライバー関連改良

従来のプログラムと設定方法

igel

プログラム

```
static struct pci_driver ehci_driver = {
    .name          = driver_name,
    .longname      = driver_longname,
    .id = { PCI_ID_ANY, PCI_ID_ANY_MASK },
    .class         = { 0x0C0320, 0xFFFFFF },
    .new           = ehci_new,
    .config_read   = ehci_config_read,
    .config_write  = ehci_config_write,
};

初期化処理 ()
{
        if (config.vmm.driver.usb.ehci)
                pci_register_driver (&ehci_driver);
}
```

設定 (bitvisor.conf)

```
vmm.driver.usb.ehci=1
```

設定 (defconfig)

```
struct config_data config = {
    .vmm = {
        .driver = {
            .usb = {
                .ehci = 1,
            },
        },
    },
};
```

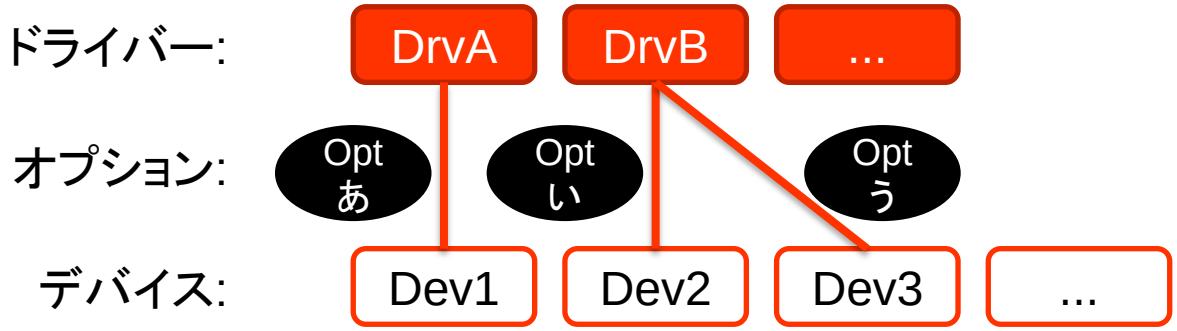
これまでの問題点と要望

igel

- 複数のIntel GbE NICのうちひとつだけをフックできない
  - 指定したデバイスのみを有効にできるようにしたい
- 新たなデバイスIDが追加された場合、同じドライバーが使えるとしても、プログラムを変更する必要がある
  - プログラムを変更することなく、ユーザーが強制的にドライバーを選択できるようにしたい
- vmm.tty_pro1000=1とすると、いずれかのIntel GbE NICからログ出力される
  - 指定したデバイスからログ出力できるようにしたい
- ドライバーと、ドライバーがゲストOSに提供する機能が一体となっている (ATA=暗号化、NIC=VPNなど)
  - ドライバーが提供する機能を選択できるようにしたい

解決策

- PCIデバイスと、それに対してどのドライバーを使用するかを、文字列で指定する
  - 指定したデバイスのみ有効にできる
  - プログラムを変更することなく、ユーザーが強制的にドライバーを選択できる
- ドライバーに対して、オプションを指定できるようにする
  - 指定したデバイスからログ出力できる
  - ドライバーが提供する機能を選択できる



デバイスの指定方法:
デバイスを選択する文字列

igel

- vmm.driver.pci_concealで使っていた文法を拡張したもの
- 以下のような項目を , (comma) で区切る。すべての条件を同時に満たすデバイスが選択される。大文字・小文字は区別される。
  - slot=%02x:%02x.%u
  - class=%04x
  - (以下省略。次ページですべての項目を挙げる。)
- 右辺にはワイルドカードが使用可能。?は任意の文字、*は0文字以上の任意の文字を意味する。また、|でOR指定も可能。

デバイスの指定方法: 項目一覧

書式 (printf形式)	引数
slot=%02x:%02x.%u	bus_no, device_no, func_no
class=%04x	class_code >> 8
id=%04x:%04x	vendor_id, device_id
subsystem=%04x:%04x	sub_vendor_id, sub_device_id
revision=%02x	revision_id
rev=%02x	revision_id
programming_interface=%02x	programming_interface
if=%02x	programming_interface
class_code=%06x	class_code
header_type=%02x	header_type
device=%s	**device name**
number=%d	**number (0, 1, 2, ...)**

プログラムの比較

1.4以前

```
static struct pci_driver ehci_driver = { (略)
  .name = "ehci_generic_driver",
  .id={PCI_ID_ANY,PCI_ID_ANY_MASK},
  .class = { 0x0C0320, 0xFFFFFF },
};
if (config.vmm.driver.usb.ehci)
  pci_register_driver (&ehci_driver);
```

- **ドライバー**: Configurationをチェックしてpci_register_driver()を呼び出す
- **PCI共通処理**: IDが一致するデバイスが見つかればドライバーを割り当てる

1.5(仮)以降

```
static struct pci_driver ehci_driver = { (略)
  .name = "ehci",
  .device = "class_code=0c0320",
};


pci_register_driver (&ehci_driver);
```

- **ドライバー**: pci_register_driver()を必ず呼び出す
- **PCI共通処理**: Configurationに沿ってドライバーを割り当てる

※ nameはデバイス名の解釈に使用。ドライバー名=デバイス名

プログラムでのオプション受け取りとデバイスID指定例

igel

- **オプション**

```
static struct pci_driver pro1000_driver = { (中略)
  .driver_options = "tty,net",
};
  char *option_net = pci_device->driver_options[1];
```

(指定がなければNULL、指定があればchar *の文字列)

- **複数デバイスID指定**

```
.device = "class_code=020000, "
          "id=8086:105e|8086:1081|8086:1082|..."
.device = "id=8086:*, class_code=030000"
```

Configurationの比較

1.4以前

bitvisor.conf

```
vmm.driver.ata=1
vmm.driver.usb.ehci=1
vmm.driver.vpn.PRO1000=1
vmm.driver.pci_conceal=class=02*,
    action=deny
```

defconfig

```
struct config_data config = { .vmm = {
    .driver = {
        .ata = 1,
        .usb = { .ehci = 1, },
        .vpn = { .PRO1000 = 1, },
        .pci_conceal="class=02*,action=deny",
    },
}, };
```

1.5(仮)以降

bitvisor.conf

```
vmm.driver.pci=driver=ata, and,
   driver=ehci, and,
   driver=pro1000, net=vpn, and,
   class=02*, driver=conceal
```

defconfig

```
struct config_data config = { .vmm = {
    .driver = {
        .pci = "driver=ata, and,"
               "driver=ehci, and,"
               "driver=pro1000, net=vpn, and,"
               "class=02*, driver=conceal",
    },
}, };
```

vmm.driver.pciの文法の概要

デバイス選択, ドライバー選択, オプション, and, ...

- **デバイス選択**: デバイスを選択する文字列。省略するとドライバー選択で指定されているデバイスが選択される。
- **ドライバー選択**: ドライバー名。必須。noneを指定するとドライバーが使用されない。
- **オプション**: ドライバーのオプション。省略可能。
- **and**: 続けて別の指定を行う場合に必須。

デバイス選択	ドライバー選択	オプション	and
	driver=ata,		and,
class=02*,	driver=conceal,		and,
	driver=pro100,	net=vpn,	and,
device=pro1000, number=0,	driver=pro1000,	net=vpn, tty=1	

vmm.driver.pciで新たにできるようになる指定の例

- 最初に見つかったIntel GbE NICのみVPNを有効にする
  - device=pro1000, number=0, driver=pro1000, net=vpn
- 00:1f.2のデバイスに対して強制的にATAドライバーを選択する
  - slot=00:1f.2, driver=ata
- 03:00.0のIntel GbE NICをゲストOSに見せずにログ出力に使う
  - slot=03:00.0, device=pro1000, driver=pro1000, tty=1
- 03:00.0のIntel GbE NICをゲストOSに見せながらログ出力にも使う (VPNモジュール不使用)
  - slot=03:00.0, device=pro1000, driver=pro1000, net=pass, tty=1

デバイス一覧・ドライバー割り当て確認機能

- make configでCONFIG_DUMP_PCI_DEV_LIST=1としてmakeすると、ドライバー初期化後に、デバイスの一覧とドライバーの割り当て状況が表示される

```
DUMP_PCI_DEV_LIST:
[00:00.0] 060000: 8086:2A40
[00:02.0] 030000: 8086:2A42 (vga_intel)
[00:02.1] 038000: 8086:2A43
[00:03.0] 078000: 8086:2A44
[00:19.0] 020000: 8086:10F5 (pro1000) pro1000
[00:1A.0] 0C0300: 8086:2937 (uhci)
[00:1A.1] 0C0300: 8086:2938 (uhci)
[00:1A.2] 0C0300: 8086:2939 (uhci)
[00:1A.7] 0C0320: 8086:293C (ehci_conceal) (ehci)
-- more --  q:continue  g:go to 1st line  r:reboot  SPC:next page  CR:next line
```

デバイス一覧・ドライバー割り当て確認機能 (続き)

```
[00:1B.0] 040300: 8086:293E
[00:1C.0] 060400: 8086:2940
[00:1C.1] 060400: 8086:2942
[00:1C.2] 060400: 8086:2944
[00:1C.3] 060400: 8086:2946
[00:1D.0] 0C0300: 8086:2934 (uhci)
[00:1D.1] 0C0300: 8086:2935 (uhci)
[00:1D.2] 0C0300: 8086:2936 (uhci)
[00:1D.7] 0C0320: 8086:293A (ehci_conceal) (ehci)
[00:1E.0] 060401: 8086:2448
[00:1F.0] 060100: 8086:2917
[00:1F.2] 010601: 8086:2929 (ahci) ahci
[00:1F.3] 0C0500: 8086:2930
[03:00.0] 028000: 8086:4236
-- more --  q:continue  g:go to 1st line  r:reboot  SPC:next page  CR:next line
```

その他のPCIドライバー関連改良

以下の共通部分をまとめた。

- **隠ぺい処理**
  一部のドライバーは、ゲストOSに見せないようにする処理をそれぞれで実装していたが、共通の実装が使えるようにした。

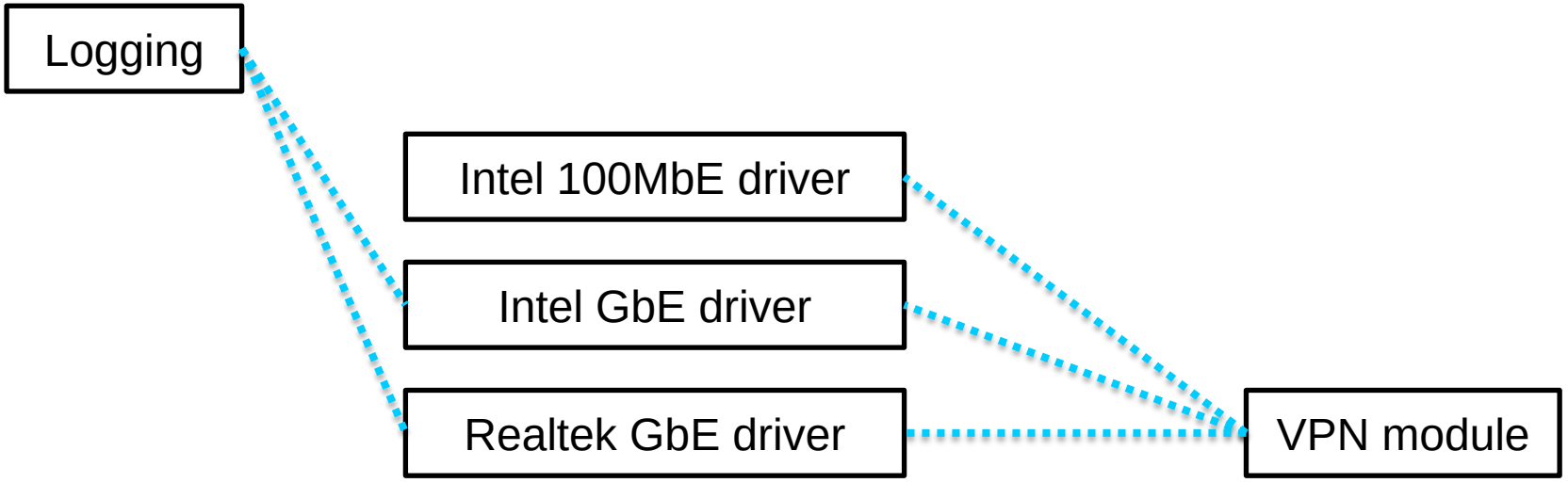
- **BAR (Base Address Register) 取得**
  ドライバーは、BARアクセスを独自に処理していたが、共通化するとともに、64bitアドレスにも対応した。

TCP/IPスタック

TCP/IPスタック導入にあたって

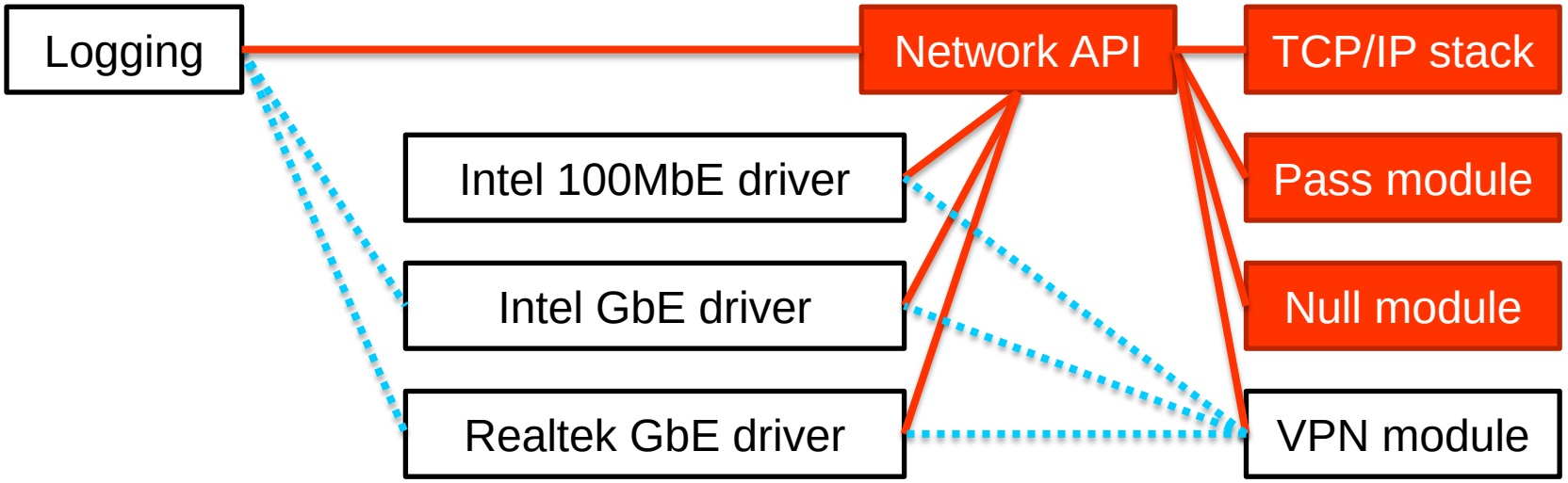
igel

- BitVisorのNICドライバーがVPNモジュールを前提としている点が問題となる
  - 後から追加されたログ出力機能も、NICドライバーごとに裏口を作って対応しなければならない状態となっている



ネットワークAPI実装

- ネットワークAPIを作り、NICドライバーとVPNなどのモジュールを結びつける役目を持たせて、機能を追加しやすくした



lwIP

igel

オープンソースの組み込み向けTCP/IPスタック

- ベアメタル用のNOSYSモードであれば組み込みは簡単
  - 時刻を提供するsys_now() 関数を実装する
  - sys_check_timeouts() 関数を頻繁に呼び出す
  - Raw APIを使用してTCP/IP通信を行うようにする
  - すべてをひとつのスレッドから実行する
- NOSYSモードを選択する理由
  - NOSYSモードにしなければ、マルチスレッド対応・Socket API対応にはなるが、いろいろ制約はあるし、通信速度は変わらない
  - Socket APIといっても、BitVisorはPOSIX互換ではないし、既存のプログラムが簡単に移植できるわけではない
  - NOSYSモードのほうが組み込みが簡単

TCP/IPスタックの使用方法

igel

- make config
  - CONFIG_IP=1
- bitvisor.conf / defconfig
  - ip.ipaddr=*192.168.12.11*
  - ip.netmask=*255.255.255.0*
  - ip.gateway=*192.168.12.254*
  - ip.use_dhcp=*0*
- NICドライバーオプション
  - ゲストOSにNICを見せない: net=ip
  - ゲストOSにNICを見せる: net=ippass

  ※今のところひとつのNICしか使用できない

  ※VMMとゲストOSの両方でDHCPを使うのは避ける

TCP/IPスタックの使用例: Echo サーバー・クライアント

igel

- サーバー: ip/echo-server.c
- クライアント: ip/echo-client.c
- 制御: ip/echoctl.c
- ユーザーインターフェイス: process/echoctl.c
  - dbgshからechoctlコマンドを実行して制御する
- ポイント
  - lwIPの呼び出しは専用のスレッドから行わなければならない
  - tcpip_begin() 関数を用いてコールバック関数のポインターを登録すると、その関数が後で専用のスレッドから呼び出されるので、そこでlwIPの呼び出しを行う
  - lwIPのAPIとBitVisorのAPIが衝突するので、lwIPの呼び出しを行うプログラムはファイルをわけておく

TCP/IPスタックの動作確認方法 (Ping)

1. **IPアドレスの確認 (DHCPの場合)**
   dbgshなどを利用してログを確認する。以下のような出力でIPアドレスが確認できる。
   IP address changed: 0.0.0.0 -> *192.168.12.11*
2. **他のホストからpingコマンドを実行**

```
$ ping 192.168.12.11
PING 192.168.12.11 (192.168.12.11) 56(84) bytes of data.
64 bytes from 192.168.12.11: icmp_seq=1 ttl=255 time=0.107 ms
64 bytes from 192.168.12.11: icmp_seq=2 ttl=255 time=0.125 ms
64 bytes from 192.168.12.11: icmp_seq=3 ttl=255 time=0.125 ms
64 bytes from 192.168.12.11: icmp_seq=4 ttl=255 time=0.250 ms
64 bytes from 192.168.12.11: icmp_seq=5 ttl=255 time=0.120 ms
...
```

TCP/IPスタックの使用例の動作確認方法 (Echoサーバー)

dbgshを使用 (CONFIG_DBGSH=1, vmm.dbgsh=1)

1. dbgshから以下のようにコマンドを実行する

```
> echoctl
echoctl> help
usage:
client connect <ipaddr> <port>  Connect to echo server.
client send                     Send a message to client.
server start <port>             Start echo server.
echoctl> server start 1234
Starting server (Port:1234)...
Done.
echoctl>
```

2. 外部からnetcatやtelnetコマンドで接続して確認する

TCP/IPスタックの使用例の動作確認方法 (Echoクライアント)

dbgshを使用 (CONFIG_DBGSH=1, vmm.dbgsh=1)

1. 適当なechoサーバーを開始するか、netcatなどで代替サーバーを準備する
2. dbgshから以下のようにコマンドを実行する

```
> echoctl
echoctl> client connect 192.168.12.12 1234
Connecting to server at 192.168.12.12:1234 (c0a80c0c)
Done.
echoctl> client send
Sending a message...
Done.
```

3. ログを見て確認する

TCP/IPスタックの通信性能 (参考)

igel

- パケットがNICに届いてから、BitVisorによって受信されるまでの時間の違いが性能に大きく影響する
  - ippass: ゲストOSにNICを見せるため、ゲストOSの割り込み処理のタイミングで受信される
  - ip: ゲストOSにNICを見せないため、その他の割り込み等でBitVisorに処理が移った時に受信される
  - ip,cpuid: ゲストOS上でひたすらCPUID命令を実行することで、頻繁にBitVisorに処理が移るため性能が向上する

種類	送信	受信
ippass	104Mbps	173Mbps
ip	7Mbps	8Mbps～176Mbps
ip,cpuid	122Mbps	204Mbps

性能改善

性能改善概要

igel

- VM Entry/Exitにかかる時間の短縮
- VM Entry/Exitの回数の削減
- Thread性能
- その他

VM Entry/Exitにかかる時間の短縮 [背景]

igel

- 某氏からの指摘「CPUID命令の実行がKVMより遅い!」
  ※Intel VT-x環境

- CPUID命令とは
  - CPUの情報を取得する命令
  - Intel VT-xにおいてVM上で実行されれば、必ずVM Exitする
  - BitVisorでは、CPUの一部機能を隠す処理を行うのみで、CPUIDによるVM Exitは極めて軽い処理のひとつである

VM Entry/Exitにかかる時間の短縮 [CPUID実行時間の測定]

igel

単位はサイクル数

- VMMなし: 81
- BitVisor: 2,887
- BitVisorの処理の単純化 (CPUIDならば何もせず直ちにVMへ戻るように): **2,449**
- KVM: 約**1,700**

```
main () {
        int a, b, c, d, min = -1;
        for (;;) {
                asm volatile ("rdtsc" : "=a" (a), "=d" (b));
                asm volatile ("xor %%eax,%%eax;cpuid"
                : : : "cc", "%rax", "%rbx", "%rcx", "%rdx");
                asm volatile ("rdtsc" : "=a" (c), "=d" (d));
                if (min < 0 || c - a < min) {
                        min = c - a;
                        printf ("%d¥n", c - a);
                }
        }
}
```

結果: VM Entry/Exitそのものに時間がかかっている!

VM Entry/Exitにかかる時間の短縮 [原因と修正]

igel

原因: VT-xのMSR (Model-specific register) 切り替え機能

- VM Entry/ExitでMSRのストア/ロードを行う機能がある
  - VMCS_VMENTRY_MSRLOAD_ADDR/COUNT
  - VMCS_VMEXIT_MSRSTORE_ADDR/COUNT
  - VMCS_VMEXIT_MSRLOAD_ADDR/COUNT
- プロセス (保護ドメイン) 機能のために、少なくとも5個のMSRを切り替えていた
- 1個減らすとCPUIDの実行時間が**300サイクル**程度短縮
- プロセス機能で必要な時にだけ切り替えるように変更して5個削減したところ、CPUIDの実行時間は1,070サイクルに!

VM Entry/Exitの回数の削減

igel

筑波大学 表さんからのコントリビューション/指摘により改良

- MSRビットマップの使用 (Intel VT-x/AMD-V)
  - 監視する必要がないMSRアクセスをフックしないようにした
- CR0/CR4 guest/host maskの使用 (Intel VT-x)
  - CR0/CR4の一部ビットの変更をフックしないようにした
  - AMD-Vは適用不可 (但しRVI使用時はCR4をフックしていない)
- CR3-load/store exitingの使用 (Intel VT-x)
  - EPT使用時はCR3の読み書きをフックしないようにした

※AMD-Vではリアルアドレスモードの処理が単純なため、RVI使用時は最初からCR3の読み書きをフックしないようにしてあった

Thread性能

igel

- 筑波大学 表さんからの指摘「スレッドスケジューリングでオーバヘッドが大きい」
- 1.4での改良 (表さんからのコントリビューション)
  - タイマー用スレッドを必要になるまで開始しないようにした
- 1.5(仮)での改良
  - schedule() をロックフリー化した
  - ついでに以下のようなコードの動作も改善した

```
if (!done) {
        do
                schedule ();
        while (!done);
}
```

メインスレッドでschedule() を呼び出し続けると、schedule() 内のspinlockを他の論理CPUが長時間 (10秒以上) ロックできなくなる場合があった

その他

igel

- MSR読み書き命令処理高速化 (AMD-V)
  - RDMSR/WRMSRはほとんどの場合2バイトの命令だが、不必要なprefix (セグメントオーバーライドなど) があると、2バイトより長くなってしまう。それに対応するためインタープリターを使用していたが、VMCBのnRIPが使用可能なCPUではインタープリターを使わないようにした。
  - Intel VT-xではすべてのCPUで命令長をVMCSから得ることができるためインタープリターは使用していない。
- インラインアセンブリでのcmpxchgの高速化
  - cmpxchg命令の実行後、ゼロフラグを得るのに、条件ジャンプを用いる代わりにsetne命令を使用するようにした。

デバッグ用新機能

デバッグ用新機能概要

- IEEE1394ログ出力
- syslogへのログ出力
- vmmstatusにVT-xのexit reasonの回数を出力
- dbgshおよびvmmstatusの64bit Windows用バイナリ生成対応

IEEE1394ログ出力

- 筑波大学 表さんからのコントリビューション
- make config
  - CONFIG_LOG_TO_IEEE1394=1
- PCIドライバー設定
  - driver=ieee1394log
- 起動時の以下のメッセージに沿って、tools/ieee1394log/にあるツールをIEEE1394の対向Linuxホストで実行する

```
================================================
   Execute 'ieee1394log 0x%llx' on host.
      Then, it will snoop:
         Phys. address: 0x%llx-0x%llx
         Virt. address: 0x%llx-0x%llx
================================================
VMM will resume boot process in 10 seconds.
```

syslogへのログ出力 [送信側]

- UDPで外部ホストのsyslogdへログ出力する機能
  - 改行文字 (LF) または80バイトごとに出力する
- bitvisor.conf / defconfig
  - vmm.tty_mac_address=(送信先MACアドレス)
  - vmm.tty_syslog.enable=1
  - vmm.tty_syslog.src_ipaddr=(送信元IPv4アドレス)
  - vmm.tty_syslog.dst_ipaddr=(送信先IPv4アドレス)
  - ルーターを超える場合は送信先MACアドレスにルーターのMACアドレスを指定する
- NICドライバーオプション
  - tty=1

syslogへのログ出力 [受信側]

Debian / Ubuntuのrsyslogd設定例

```
$ cat /etc/rsyslog.d/10-bitvisor.conf
$FileCreateMode 0644
:syslogtag,isequal,"bitvisor:" /var/log/bitvisor.log
& ~
```

別ファイルに出力

```
$ grep -i udp /etc/rsyslog.conf
# provides UDP syslog reception
$ModLoad imudp
$UDPServerRun 514
```

UDP有効

受信例

```
Nov  7 09:44:02 10.16.149.126 bitvisor: Processor 1 (AP)
Nov  7 09:44:02 10.16.149.126 bitvisor: Processor 2 (AP)
Nov  7 09:44:02 10.16.149.126 bitvisor: Processor 3 (AP)
Nov  7 09:44:02 10.16.149.126 bitvisor: Trying to enable VMXON.
Nov  7 09:44:02 10.16.149.126 bitvisor: message repeated 2 times: [Trying to enable VMXON.]
Nov  7 09:44:02 10.16.149.126 bitvisor: Processor 1 2693827920 Hz (Invariant TSC)
Nov  7 09:44:02 10.16.149.126 bitvisor: Processor 2 2693825904 Hz (Invariant TSC)
Nov  7 09:44:02 10.16.149.126 bitvisor: Processor 3 2693829552 Hz (Invariant TSC)
Nov  7 09:44:03 10.16.149.126 bitvisor: AHCI 0:0 IDENTIFY
```

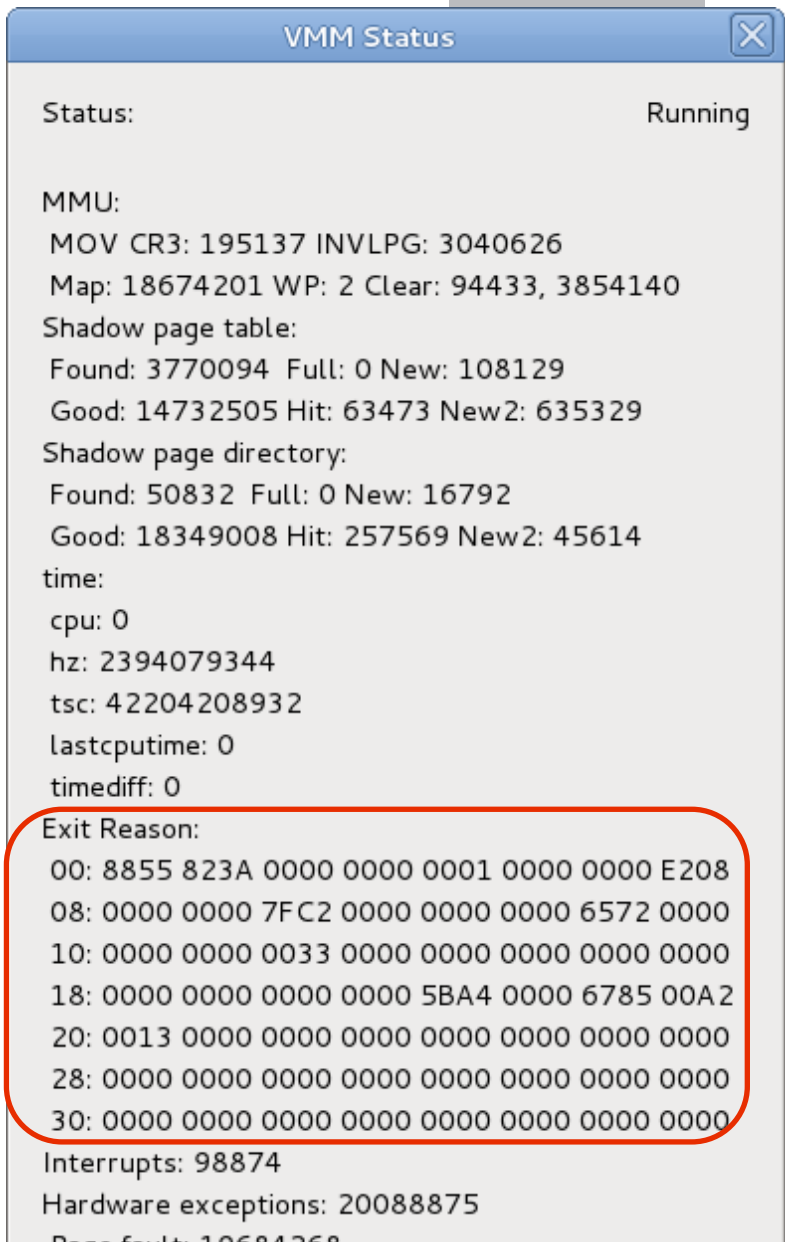
vmmstatusにVT-xのexit reasonの回数を出力

vmmstatus: VMMのステータスを読み取って表示するプログラム

- GTK+版: tools/vmmstatus-gtk
- Windows版: tools/vmmstatus-win32
- CONFIG_STATUS=1
- vmm.status=1

Exit reasonの回数を出力

- 回数の下位16ビット
- デバッグや性能改善の目安に



dbgshおよびvmmstatusの64bit Windows用バイナリ生成対応

igel

- 64bitバイナリの必要性
  - 通常は64bit Windowsでも32bitバイナリを使用すればよい
  - WindowsインストールDVDなどで使われているWindows PE環境では、32bitバイナリが実行できない場合があるので、64bitバイナリもあると便利
- 64bit対応の変更内容
  - unsigned longをintptr_tに (64bit Windowsではlongが32bit)
  - DialogProcの返り値をBOOLからINT_PTRに
  - SetWindowLongをSetWindowLongPtrに
  - STDCALLをWINAPIに
- 64bitコンパイル方法 (Debian環境)
  - make MINGW_PREFIX=amd64-mingw32msvc-

その他バグ修正など

その他の修正内容 (1)

igel

- Intel X540 (10GbE NIC) driver (筑波大学 表さんからのコントリビューション)
  - 現状はログ出力専用
- acpiのコンパイルエラー修正 (筑波大学 表さんからのコントリビューション)
- tools/vmmstatus-gtkのタイマーが動かないバグの修正
- pro1000の受信時に見るディスクリプターが正しくなかったかもしれない記述ミスの修正
- write_gphys_qがページ境界をまたぐと正しく書き込まないバグの修正
- tools/logのLinux 2.6.37以降対応

その他の修正内容 (2)

igel

- 起動後1時間11分35秒過ぎると、ahciのVMMからのコマンド発行が必ずタイムアウトしてしまうバグの修正
  - 1時間11分35秒≒2^{32}マイクロ秒=4,294,967,296マイクロ秒
- GNU Make 4.0対応
  - BSD Make対応のために、GNU Makeが対応していない文法を使用して識別していたが、GNU Make 4.0がその文法に対応してしまったことでmakeできなくなっていた
- CONFIG_ENABLE_ASSERT=0でコンパイルすると正常に動かない問題を解決する仕様変更
  - ASSERTの引数を常に評価するように変更
- initfuncの並べ替えでファイル名もキーとするように変更
  - 並べ替えの順番で起きていた問題の解決

まとめ

igel

BitVisor 1.5(仮)で新しくなること

- PCIデバイスドライバー関連
- TCP/IPスタック
- 性能改善
- デバッグ用新機能
- その他バグ修正など

BitVisor 1.5(仮)リリース予定日: 近々発表できると思う