デバイス隠蔽によるPCIデバイス分割(改版)

2013年3月10日
乃村研究室 池田 騰

1 はじめに

本資料は，資料⟨No.245-08⟩の改版である．変更点としては，具体的な実装方法を追加している．具体的な内容は以下の通りである．Mintにおいてデバイスを隠蔽することによってOSノード間のデバイス分割を行う方法について検討する．現在のMintでは，デバイスの分割を占有しないデバイスのデバイスドライバとの対応付けを制限することによって実装している．この方法により，デバイスドライバを利用したデバイスの操作は防げる．一方で，OSノードは通常のデバイスと同じように占有しないデバイスを認識するため，デバイスドライバを経由しない占有しないデバイスへの操作は可能となっている．OSノード間の独立性を高めるため，占有しないデバイスをOSノードから隠蔽し，資源として認識させないデバイス分割法を実装する．

2 OSによるデバイス操作

OSによるデバイス操作の例を図1に示し，以下で説明する．OSによるデバイス操作には以下の2つの経路がある．

(1) OSによる直接のデバイス操作

OSがデバイスドライバを経由せずに直接デバイスを操作する場合である．デバイスの検出や登録の際には，OSはデバイスドライバを経由せずに，I/Oアドレス空間を直接操作してデバイスの操作を行う．

(2) デバイスドライバを経由したデバイス操作

一般的にデバイスを利用する場合，そのデバイスに対応するデバイスドライバがデバイスの操作を行う．この際，デバイスドライバは，(1)と同じく，OSの提供するI/Oアドレス空間のインタフェースを用いてデバイスの操作を行う．

上記のように，Linuxにおけるデバイス操作は必ずOSの提供するI/Oアドレス空間を用いたデバイス操作インタフェースを利用する．このため，ここでデバイスの使用を制限することで(1)と(2)の双方によるデバイス操作を制限できる．

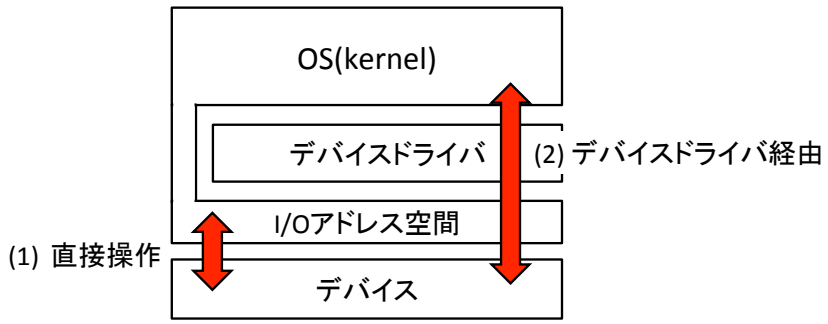

図 1: デバイス操作

3 デバイス分割方式

Mint において，デバイスを分割する方式として，以下の 2 つが存在する．

(方式 1) 占有しないデバイスに対してデバイスドライバを使用した操作のみを防ぐ方式

各 OS ノードに全ての資源を認識させておき，他占有資源に関しての占有のみを防ぐ方式である．図 1 における (2) を防ぐ．現在の Mint ではこの方式でデバイス分割を行っている．図 2 にデバイス初期化までの流れを示し，デバイス分割方法について説明する．この方式では，全てのデバイスの検出と登録が修了した後に，占有しないデバイスはデバイスドライバの対応付けを防ぐことによってデバイスの初期化と利用を制限している．このため，OS ノードは占有しないデバイスについても通常と同じように認識しており，lspci などのデバイスドライバを経由しないデバイス操作は占有しないデバイスに関しても利用可能となってしまう．

(方式 2) OS ノードから自占有資源以外を完全に隠蔽する方式

各 OS ノードから，占有デバイス以外を完全に隠蔽し，認識できない状態にする方式である．図 1 における (1) と (2) を両方防ぐことが可能である．この方式は，Linux によるデバイス操作部分を改変して実装する必要がある．(方式 1) と比較して，OS ノード間の独立性が高いという利点がある．これは，占有しないデバイスは資源として認識させないため，どのようなソフトウェアからも操作不可能であるためである．

OS ノード間の独立性を考慮し，(方式 2) を実現する．この際，Linux によるデバイス操作部分を改変する必要があるため．PCI デバイスの操作について次章で説明する．PCI デバイスに限定するのは，現在のデバイス実装では，PCI デバイスがデ

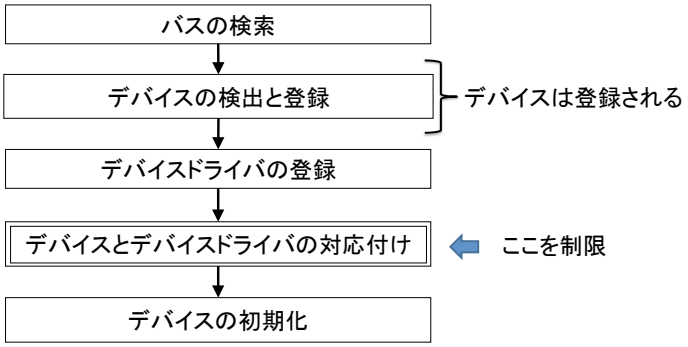

図 2: デバイス初期化までの流れ

ファクト・スタンダードとなっており，今後は PCI express としてデバイスが実装されていくと予想されるためである．

4 Linux における PCI デバイス操作

4.1 基本構造

Linux による PCI デバイス操作は I/O アドレス空間へのアクセスとして実装されている．具体的には，PCI デバイス操作用の領域が I/O アドレス空間に用意されており，その領域を操作することによって PCI デバイスを操作可能である．以下で，PCI デバイス操作の基本構造として，PCI デバイス操作用の I/O アドレス空間，PCI コンフィグレーション空間，および CONFIG_ADDRESS について説明する．

(1) PCI デバイス操作用の I/O アドレス空間

Linux では I/O アドレス空間内に I/O 値を読み書きすることによって，デバイスを制御している．図 3 に I/O アドレス空間を示し，以下で説明する．

I/O アドレス空間には，全 PCI デバイスを管理するための領域が存在する．この領域は 0xCF8-0xCFF であり，前半部分の 0xCF8-0xCFB と後半部分の 0xCFC-0xCFF の領域に分けられる．以下でそれぞれの領域について示す．

前半領域 (0xCF8-0xCFB): 書き込み専用の領域である．CONFIG_ADDRESS を書き込むことでどのデバイスの PCI のコンフィグレーション空間のどのレジスタにアクセスするかを決定する．

後半領域 (0xCFC-0xCFF): PCI コンフィグレーション空間の書き込み/読み込み用の領域である．前半部分の領域で指定したデバイスの PCI コン

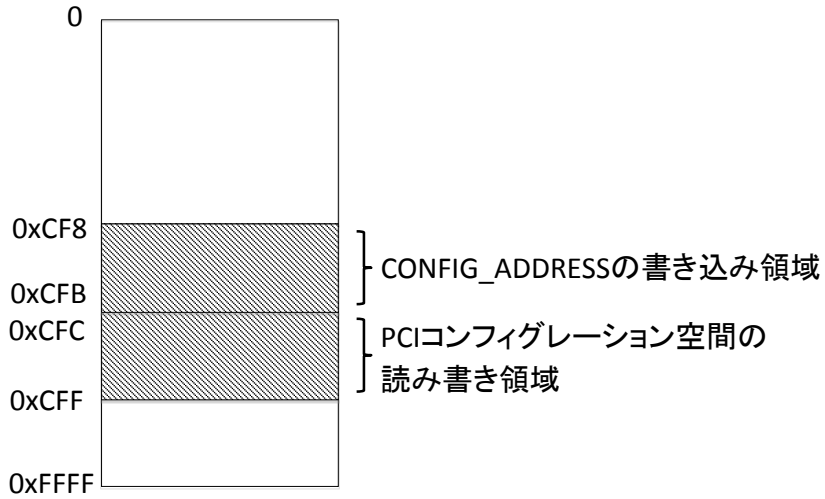

図 3: I/O アドレス空間

フィグレーション空間のレジスタがマッピングされる．この領域を読み書きすることでデバイスを制御する．

(2) PCI コンフィグレーション空間

全ての PCI デバイスが持つ PCI デバイスのレジスタにアクセスするための空間である．図 4 に一般的な PCI デバイスの PCI コンフィグレーション空間を示し，以下で説明する．PCI コンフィグレーション空間には，PCI デバイスを操作するためのレジスタが割り当てられており，各レジスタを読み書きすることによってデバイスの操作が可能となっている．具体的には，以下のようなレジスタを持っている．

(A) ベンダ ID，デバイス ID
デバイスの識別に使用する ID である．このベンダ ID とデバイス ID はデバイスの工場出荷時に設定された値であり，ベンダ ID は製造元企業を示し，デバイス ID は製造元企業が割り当てたデバイスの番号を示す．

(B) ステータス
規格上で定められた意味を持つビットによって，デバイスの状態を表す．

(C) コマンド
このフィールドに書き込むことで，システムはデバイスを制御する．

(D) クラスコード
ビデオや SCSI などすべての 種類のデバイスには規格上のクラスがあり，そのデバイスのタイプを特定するものである．

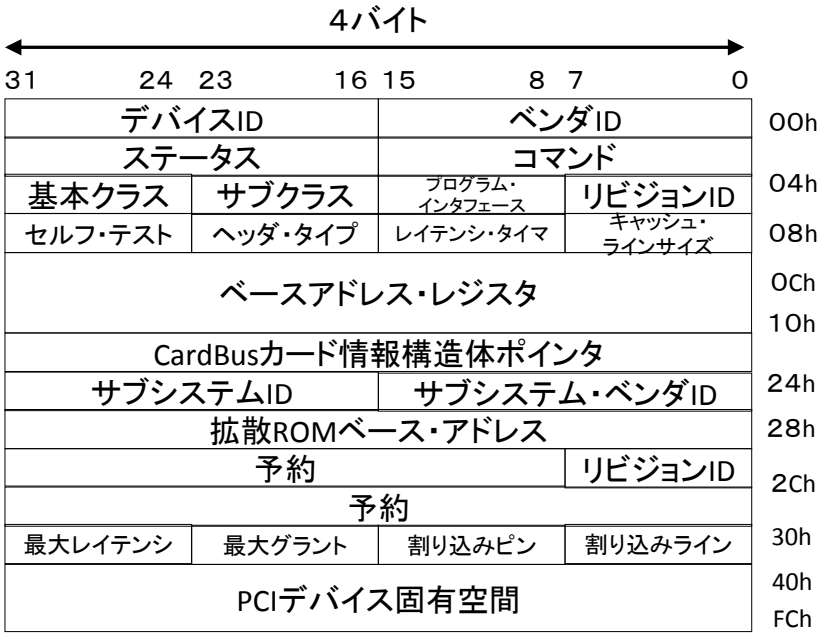

図 4: 一般的な PCI デバイスの PCI コンフィグレーション空間

(3) CONFIG_ADDRESS

CONFIG_ADDRESS のフォーマットを図5 に示し，以下で説明する．OS から見て対象のデバイスがどのバスの何番目のデバイスであるかなどの情報をバス番号やデバイス番号として格納されている．OS の認識しているデバイスの PCI コンフィグレーション空間とレジスタの指定に使用する．

具体的なフィールドは以下の通りである．

(A) バス番号
物理的にデバイスが計算機のどのバスに設置されているかを示す番号である．

(B) デバイス番号
バス上の各スロットに付けられた番号である．

(C) ファンクション番号
デバイスの持つ機能の番号である．

(D) レジスタ番号
操作デバイスの持つどのレジスタにアクセスするかを示す番号である．

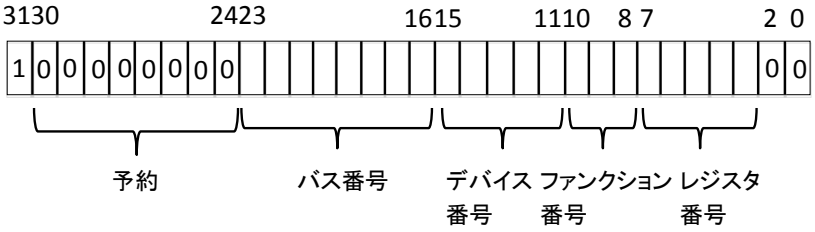

図 5: CONFIG_ADDRESS

4.2 PCI コンフィグレーション空間操作関数

PCI コンフィグレーション空間の読み書きには特定の関数が使用される．このため，この関数を改変することによって，全ての PCI デバイスへの操作を改変できる．

具体的には，arch/x86/pci/direct.c というファイルで定義されている pci_conf1_read という関数により読み込みを行い，pci_conf1_write という関数で書き込みを行う．

これらの関数は以下の処理を行っている．

(1) I/O アドレス空間の 0xFC8 に CONFIG_ADDRESS を書き込む

I/O アドレス空間の CONFIG_ADDRESS 書き込み専用領域 (0xCF8-0xCFB) に CONFIG_ADDRESS を書き込む．これにより，操作したいデバイスの PCI コンフィグレーション空間とレジスタを指定する．

(2) I/O アドレス空間の 0xFCF に書き込み/読み込み命令を行う

0xFC8 に書きこまれた CONFIG_ADDRESS に対応する PCI コンフィグレーション空間のレジスタを読み書きする．

4.3 PCI デバイス検出の流れ

占有しないデバイスが OS ノードに登録されるのを防ぐためには，デバイスの登録処理を改変する必要がある．そこで，PCI デバイスが検出され，登録されるまでの流れを示し，以下で説明する．

(1) バス 0 をスキャンし，バス 0 上の全てのスロットの PCI デバイスの PCI コンフィグレーション空間を確認し，ベンダ ID とデバイス ID を読み出す．

(2) ベンダ ID とデバイス ID をもとにデバイスが存在するか否かを判断する．なお，何も刺さっていないスロットのベンダ ID とデバイス ID はそれぞれ 0xffff となっている．

(A) デバイスが存在する場合は，そのPCIデバイスの構造体を作成し，情報を書き込む．また，全PCIデバイスのリストに追加する．

(B) 発見したデバイスがブリッジ場合である場合は，バスの構造体を作成し，情報を書き込む．また，全PCIバスのリストに登録したバスを追加する．

(3) ブリッジが発見された場合，ブリッジ以下のバスに対して (1), (2) の処理を繰り返す．

上記 (2) において，占有しないデバイスを空スロットのデバイスに見せかけることにより，OSノードが占有しないデバイスを登録することを防げる．

5 デバイス隠蔽によるデバイス分割

Mintにおいて，デバイスの隠蔽によるデバイス分割を行うためには，以下の2つの課題が存在する．

(課題1) OSノードから占有しないデバイスを隠蔽

OSノードがデバイスの情報を登録する前にデバイスを隠蔽することにより，占有しないデバイスに対してOSノードが影響を与えることを防ぐ．また，占有していないデバイスを隠蔽することにより，Linux の既存機能によりOSノード内の占有デバイスのみを管理することも可能となる．

(課題2) 動的なデバイスの隠蔽と検出

OSノード間でのデバイス移譲を考慮すると，OSノードから隠蔽しているデバイスを動的に利用可能にできる必要がある．このため，占有状態の変化に伴い，動的にデバイスの隠蔽と検出を可能にする必要がある．

これらの課題についての対処をそれぞれ以下に示す．

(対処1) PCIコンフィグレーション空間へのアクセス時に占有デバイス以外への操作をフックし，空スロットへのアクセスに見せかける．

OSノードがデバイスの操作を行う場合，6章で説明した関数を必ず利用する．この関数の処理をフックし，占有していないデバイスへのアクセスを空スロットへのアクセスと同等の値を返すように変更を加える．これにより，実際には存在しているデバイスをOSノードから隠蔽することができる．

図6に，PCIコンフィグレーション空間にアクセスする関数に，デバイスを隠蔽するための処理を実装した際の処理フロー図を示し，以下で説明する．下記の手順の処理のうち，(1)-(4) が新たに追加した処理である．

(1) I/O アドレス空間 0xCF8 にデバイス確認用の CONFIG_ADDRESS を書き込む

どのデバイスを操作しようとしているのかを確認するため，書き込み予定の CONFIG_ADDRESS のレジスタフィールドを 0 にし，ベンダ ID とデバイス ID を指し示すようにしてから I/O アドレス空間 0xCF8 に書き込む.

(2) ベンダ ID とデバイス ID の読み込み

0xCFC から先頭 4 バイトを読み込み，ベンダ ID とデバイス ID を読み込む.

(3) 占有デバイスか否か

ベンダ ID とデバイス ID から占有しているデバイスか否かを判別する．占有していないデバイスであれば，(4) の処理へ移行する．占有しているデバイスであれば通常の処理である (5) に移行する.

(4) 値に 0xffffffff を格納

空スロットへのアクセスと同じ値である 0xffffffff を値として返して修了する.

(5) アドレス空間 0xCF8 にデバイス操作用の CONFIG_ADDRESS を書き込む

占有しているデバイスであるため，通常の処理通り，アドレス空間 0xFC8 に操作デバイスとレジスタを指定するための CONFIG_ADDRESS　を書き込む.

(6)　レジスタの書き込み/読み込み処理通常の処理通りの書き込み，または読み込み処理を行う.

上記の処理により，占有していないデバイスへのアクセスを空スロットへのアクセスに見せかけることができる．結果として，PCI デバイスへの 4.3 節で述べたデバイス検出時のデバイス登録も防げるため，OS ノード内に占有しないデバイスの情報が登録されることも同時に防ぐことができる.

(対処 2) 隠蔽するデバイスを変更し，Linux の既存機能により PCI バスの再スキャンを行う.

Linux には PCI デバイスのホットプラグに対応するため，PCI バスを再スキャンし，再構成する機能が存在する．この機能を用いることで，デバイス移譲の際に，動的にデバイスの隠蔽状態を変更することが可能となる．具体的には，新たに占有するデバイスをアクセス可能なデバイスのリストに追加し，PCI バスの再スキャンを行うことで，デバイスの検出と登録が行われる.

上記の対処により，Mint におけるデバイス隠蔽によるデバイス分割が可能となる.

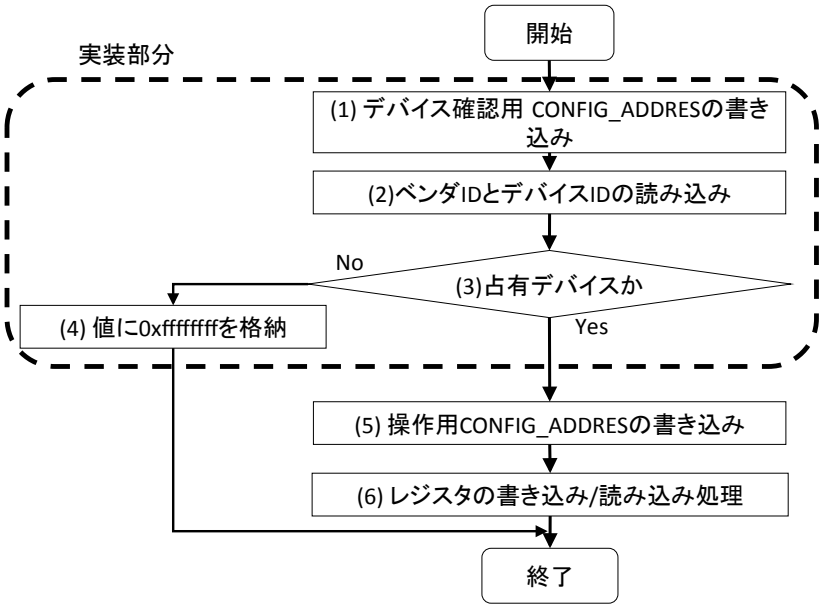

図 6: PCI コンフィグレーション空間へのアクセス関数におけるデバイスの隠蔽の実装

6 実装

6.1 デバイス情報の取得方法

本デバイス分割法では，分割するデバイスの指定にベンダ ID とデバイス ID を用いる．このため，ベンダ ID とデバイス ID の取得方法を以下で述べる．

接続されている PCI デバイスのベンダ ID とデバイス ID の確認には lspci コマンドを用いると良い．lspci のオプションのうち，デバイス特定に有用なオプションを表 1 を示す．例として，以下で NIC のベンダ ID とデバイス ID を確認する手順を示す．

(1) lspci コマンドを実行し，全ての PCI デバイス情報から目的のデバイスの Bus_id を確認

lspci コマンドを実行することで，計算機に接続されている全ての PCI デバイスの情報を得ることができる．この際，-v や-vv オプションを付けることでより詳細な情報を得ることができるため，デバイスの識別に必要な場合は使用する．

例として，以下で lspci コマンドを用いた際に出力された NIC の情報を示す．このうち，最初の行の 06:02.0 はそのデバイスの存在するバス番号とデバイス番号を組み合わせた ID(以下，Bus_id と呼ぶ) である．また，最後の行の kernel

driver in use: e1000 がそのデバイスと対応づけられているドライバ名を示している．

```
#lspci -v
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 略 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
06:02.0 Ethernet controller: Intel Corporation 82541PI Gigabit /
        Ethernet Controller (rev 05) Subsystem: 8086:1376
        Flags: bus master, 66MHz, medium devsel, latency 64, IRQ 17
        Memory at f7fe0000 (32-bit, non-prefetchable) [size=128K]
        Memory at f7fc0000 (32-bit, non-prefetchable) [size=128K]
        Expansion ROM at f7fa0000 [disabled] [size=128K]
        Capabilities: <access denied>
        Kernel driver in use: e1000
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 略 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

(2) lspci コマンドを-n コマンドを付けて実行し，ベンダ ID とデバイス ID を確認

lspci コマンドでは，ベンダ ID とデバイス ID を元に，PCI ID データベースからデバイスの情報を得て表示する．lspci コマンドに-n オプションを付けることによって，ベンダ ID とデバイスコードをデバイス情報に変換せずにそのまま表示することができる．この際，デバイス情報は表示されないため，デバイスの識別は (1) で確認したデバイス ID を用いると良い．

以下に-n オプションを付けた状態で (1) を同じ情報を表示させた結果を示す．1 行目の 06:02.2 の後に 0200: 8086:107c がデバイス情報の代わりに表示されていることが分かる．このうち，8086 がベンダ ID であり，107c がデバイス ID である．

```
#lspci -v -n
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 略 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
06:02.0 0200: 8086:107c (rev 05)
        Subsystem: 8086:1376
        Flags: bus master, 66MHz, medium devsel, latency 64, IRQ 17
        Memory at f7fe0000 (32-bit, non-prefetchable) [size=128K]
        Memory at f7fc0000 (32-bit, non-prefetchable) [size=128K]
        I/O ports at ec00 [size=64]
        Expansion ROM at f7fa0000 [disabled] [size=128K]
        Capabilities: <access denied>
        Kernel driver in use: e1000
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 略 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

表 1: lspci コマンドのオプション

-v	デバイス情報を詳細に表示
-vv	デバイス情報をより詳細に表示
-n	PCI ID データベースを調べてデバイス情報を表示する 代わりにベンダ ID とデバイスコードを表示

7 占有デバイスの判別

別紙に実験用計算機の PCI デバイスについて，ベンダ ID，デバイス ID の情報を含めた一覧を示す．また，これらのデバイスについて，OS0，OS1，および OS2 において，占有するべきか否かを示し，以下で説明する．なお，本誌料では OS0 がディスプレイとキーボードを占有し，OS1 がシリアルポートを占有し，OS2 が NIC を占有することを想定している．

OS 間で PCI デバイス分割を行うべきデバイスは以下の 8 つである．

(通番 9) USB Controller

現在の Mint では USB を使用する OS を決めていないため，今のところ全ての OS に占有させていない．USB を必要とする OS があれば，その OS に占有させる．

(通番 10) Audio device

現在の Mint ではサウンドカードを使用する OS を決めていないため，今のところ全ての OS に占有させていない．サウンドカードを必要とする OS があれば，その OS に占有させる．

(通番 15) USB Controller

(通番 1) と同様である．

(通番 18) ISA Interface

ata_host2 のデバイスである．現状の Mint では，OS0 と OS1 の HDD を ata_host2 に接続しているため，OS1 と OS2 には占有させ，OS3 には占有させない．なお，ata_host2 に接続されている OS0 と OS1 の 2 つの HDD をここで分割することはできないため，ata_piix ドライバ内部の処理を改変することで分割している．

(通番 20) ISA Interface

ata_host3 のデバイスである．OS3 のみが占有する．

(通番 21) VGA compatible Controller

ディスプレイは OS0 が占有するため，OS2 と OS3 からは隠蔽する．

(通番 22) Audio device

（通番 10）と同様である．

(通番 25) Ethernet Controller

Network card は OS3 の起動に用いるため，OS0 と OS1 からは隠蔽する．

8 PCI デバイス分割の実装

PCI デバイスを分割させるため，arch/x86/pci/direct.c の pci_conf1_read() 関数と pci_conf1_write() 関数において，占有しないデバイスについては読み込み処理と書き込み処理を行わないように対象のデバイスを認識できないように改変した．pci_conf1_read について改変したコードを以下に示し，説明する．pci_conf1_write に関しては，ほぼ同様な変更であるため，割愛する．なお，追加した行は+で示す．

```
 1  static int pci_conf1_read(unsigned int seg, unsigned int bus,
 2                            unsigned int devfn, int reg, int len, u32 *value)
 3  {
 4          unsigned long flags;
 5      int i;
 6          if ((bus > 255) || (devfn > 255) || (reg > 4095)) {
 7                  *value = -1;
 8                  return -EINVAL;
 9          }
10
11          raw_spin_lock_irqsave(&pci_config_lock, flags);
12
13  +     /*
14  +      * For Mint by IKEDA(20130306)
15  +      * Hide non occupancy PCI device for device allocation.
16  +     */
17  +        outl(PCI_CONF1_ADDRESS(bus, devfn, 0), 0xCF8);
18  +
19  +     for(i=0; i < uni_hide_devices; i++){
20  +         if(inl(0xCFC) == mint_uni_hide_device[i]){
21  +             *value = 0xffffffff;
22  +             goto out;
23  +         }
24  +     }
25  +     switch (mint_device_allocation_set) {
26  +         case MINT_DEVICE_PRE:
27  +             for(i=0; i < pre_hide_devices; i++){
28  +                 if(inl(0xCFC) == mint_pre_hide_device[i]){
29  +                     *value = 0xffffffff;
30  +                     goto out;
```

```
31 +                }
32 +            }
33 +            break;
34 +        case MINT_DEVICE_CO:
35 +            for(i=0; i < co_hide_devices; i++){
36 +                if(inl(0xCFC) == mint_co_hide_device[i]){
37 +                    *value = 0xffffffff;
38 +                    goto out;
39 +                }
40 +            }
41 +            break;
42 +        case MINT_DEVICE_TRI:
43 +            for(i=0; i < tri_hide_devices; i++){
44 +                if(inl(0xCFC) == mint_tri_hide_device[i]){
45 +                    *value = 0xffffffff;
46 +                    goto out;
47 +                }
48 +            }
49 +            break;
50 +    }
51
52      outl(PCI_CONF1_ADDRESS(bus, devfn, reg), 0xCF8);
53
54          switch (len) {
55          case 1:
56                  *value = inb(0xCFC + (reg & 3));
57                  break;
58          case 2:
59                  *value = inw(0xCFC + (reg & 2));
60                  break;
61          case 4:
62                  *value = inl(0xCFC);
63                  break;
64          }
65
66  +  out:
67          raw_spin_unlock_irqrestore(&pci_config_lock, flags);
68
69          return 0;
70  }
```

17行目でPCIコンフィグレーション空間のベンダIDとデバイスIDの領域にアクセスできるようにCONFIG_ADDRESSを指定している．19行目から40行目で各OSノードのについて占有しないデバイスか否かをチェックし，占有しないデバイスであれば66 行めのout: に飛びようにしている．

また19行目から40行目で使用している配列は各OSノードで隠蔽するデバイスを格納しており，arch/x86/include/asm/mint_devices.h で宣言している．具体的には以下のように宣言している．

```
1  int mint_uni_hide_device[]={
2  0x3b348086,    /* USB2.0 */
```

```
 3  0x3b3c8086,    /* USB2.0 */
 4  0xd1388086,    /* HDA_intel */
 5  0x3b568086,    /* Sound Card */
 6  0x0be310de     /* Audeo device controller*/
 7  };
 8  unsigned int uni_hide_devices=4;
 9
10  int mint_pre_hide_device[]={
11  0x107c8086,  /* e1000 NIC */
12  0x3b268086   /* ata_host3 */
13  };
14  unsigned int pre_hide_devices=2;
15
16  int mint_co_hide_device[]={
17  0x0a6510de, /* VGA controller */
18  0x107c8086, /* e1000 NIC */
19  0x3b268086  /* ata_host3 */
20  };
21  unsigned int co_hide_devices=3;
22
23  int mint_tri_hide_device[]={
24  0x0a6510de, /* VGA controller */
25  0x3b208086  /* ata_host2 */
26  };
27  unsigned int tri_hide_devices=2;
```

宣言している各配列はそれぞれ以下の通りである．

(1) mint_uni_hide_device[ ]: 共通で占有しないデバイスを格納する．

(2) mint_pre_hide_devices[ ]: OS ノード 0 が占有しないデバイスを格納する．

(3) mint_co_hide_devices[ ]: OS ノード 1 が占有しないデバイスを格納する．

(4) mint_tri_hide_devices[ ]: OS ノード 2 が占有しないデバイスを格納する．

これらの配列に占有しないデバイスのデバイス ID とベンダ ID を既述することで PCI デバイスの分割を行う．

9 デバイスの指定方法について

現在の実装では，占有しないデバイスをブラックリスト形式で指定している．これは，占有しないデバイスの方が数が少ないためである．一方で，占有するデバイスを指定する方が利用者が設定しやすいという利点がある．ブラックリスト形式をホワイトリスト形式にするためには，pci_conf1_read() と pci_conf1_write() を書き換えれば容易に変更できる．

10　残された課題について

(1) 動的に占有資源配列を変更可能にする．

本デバイス分割機能では，まだ配列に設定したデバイスを動的に変更する機能を作成できていない．このため，動的にデバイスを移譲ことができない．

(2) 起動時に占有するデバイスを指定可能にする．

起動時に自由に占有する資源を選択可能にすることで，より柔軟な資源の利用が可能となる．例えば，現状では起動時に占有資源をあらかじめ決定しているため，他 OS ノード起動用の資源を別の OS ノードが使用するとその OS ノードは起動することはできない．この制約を撤廃することで，その時々の資源の状況に応じた OS ノード構成での起動が可能となる．

11　まとめ

Mint におけるデバイス隠蔽による PCI デバイス分割方法について説明した．計算機に新たにデバイスが接続されると，分割するデバイスを新たに設定する必要があることに注意が必要である．また，今回実装したカーネルへの変更は TwinOS2.6 用 Git リポジトリ [1] から，以下のコミット番号で参照できる．

- 2d3db8b548a8f547d57e5a19a6a7bae4151c187b

参考文献

[1] TwinOS26 Git repository, ”http://newgroup.swlab.cs.okayama-u.ac.jp/gitweb/cgi-bin/gitweb.cgi?p=TwinOS26.git;a=summary

実験用計算機のPCIデバイス一覧

2014/3/10
Newグループ：池田騰

通番	OS0	OS1	OS2	Bus_id	ベンダID	デバイスID	ドライバ名等	備考
1	○	○	○	00:00.0	8086	d131	Host bridge	DMI
2	○	○	○	00:03.0	8086	d138	PCI bridge	PCI Express root port 1
3	○	○	○	00:08.0	8086	d155	System peripheral	System Management Registers
4	○	○	○	00:08.1	8086	d156	System peripheral	Semaphore and Scratchpad Registers
5	○	○	○	00:08.2	8086	d157	System peripheral	System Control and Status Registers
6	○	○	○	00:08.3	8086	d158	System peripheral	Miscellaneous Registers
7	○	○	○	00:10.0	8086	d150	System peripheral	QPI Link
8	○	○	○	00:10.1	8086	d151	System peripheral	QPI Routing and Protocol Registers
9	×	×	×	00:1a.0	8086	3b3c	USB Controller(ehci_hcd)	USB2 Enhanced Host Controller
10	×	×	×	00:1b.0	8086	3b56	Audio device	High Definition Audio
11	○	○	○	00:1c.0	8086	3b42	PCI bridge(pcieport)	PCI Express Root Port 1
12	○	○	○	00:1c.4	8086	3b4a	PCI bridge(pcieport)	PCI Express Root Port 5
13	○	○	○	00:1c.5	8086	3b4c	PCI bridge(pcieport)	PCI Express Root Port 6
14	○	○	○	00:1c.6	8086	3b4e	PCI bridge(pcieport)	PCI Express Root Port 7
15	×	×	×	00:1d.0	8086	3b34	USB Controller(ehci_hcd)	USB2 Enhanced Host Controller
16	○	○	○	00:1e.0	8086	244e	PCI bridge	82801 PCI bridge
17	○	○	○	00:1f.0	8086	3b02	ISA bridge	LPC Interface Controller
18	○	○	×	00:1f.2	8086	3b20	ISA Interface(ata_piix)	4 port SATA IDE Controller
19	○	○	○	00:1f.3	8086	3b30	SMBus(i801_smbus)	4 port SMBus Controller
20	×	×	○	00:1f.5	8086	3b26	ISA Interface(ata_piix)	2 port SATA IDE Controller
21	○	×	×	01:00.0	10de	0a65	VGA compatible Controller	GeForce210
22	×	×	×	01:00.1	10de	0be3	Audio device	High Definition Audio Controller
23	○	○	○	02:00.0	197b	2361	SATA controller(ahci)	JBM361 AHCI/IDE
24	○	○	○	02:00.1	197b	2361	IDE interface	JBM361 AHCI/IDE
25	×	×	○	06:02.0	8086	107c	Ethernet controller(e1000)	82541PI Gigabit Ethernet Controller
26	○	○	○	3f:00.0	8086	2c51	Host bridge	QuickPath Architecture Generic Non-Core Resisters
27	○	○	○	3f:00.1	8086	2c81	Host bridge	QuickPath Architecture System Address Decoder
28	○	○	○	3f:02.0	8086	2c90	Host bridge	QPI Link 0
29	○	○	○	3f:02.1	8086	2c91	Host bridge	QPI Physical 0
30	○	○	○	3f:03.0	8086	2c98	Host bridge	Integrate Memory Controller
31	○	○	○	3f:03.1	8086	2c99	Host bridge	Integrate Memory Controller Targer Address Decoder
32	○	○	○	3f:03.4	8086	2c9c	Host bridge	Integrate Memory Controller Test Register
33	○	○	○	3f:04.0	8086	2ca0	Host bridge	Integrate Memory Controller Channel 0 Control Registers
34	○	○	○	3f:04.1	8086	2ca1	Host bridge	Integrate Memory Controller Channel 0 Address Registers
35	○	○	○	3f:04.2	8086	2ca2	Host bridge	Integrate Memory Controller Channel 0 Rank Registers
36	○	○	○	3f:04.3	8086	2ca3	Host bridge	Integrate Memory Controller Channel 0 Thermal Control Registers
37	○	○	○	3f:05.0	8086	2ca8	Host bridge	Integrate Memory Controller Channel 1 Control Registers
38	○	○	○	3f:05.1	8086	2ca9	Host bridge	Integrate Memory Controller Channel 1 Address Registers
39	○	○	○	3f:05.2	8086	2caa	Host bridge	Integrate Memory Controller Channel 1 Rank Registers
40	○	○	○	3f:05.3	8086	2cab	Host bridge	Integrate Memory Controller Channel 1 Thermal Control Registers

○：占有
×：非占有