デバイスの占有機能(修正版3)

報告日 2009/12/22
千崎良太

1 はじめに

本資料は<No.137-06>の修正版である．
本資料では，NewOS におけるデバイスの占有方法について説明する．

2 概要

本資料で説明するデバイスの占有方法は以下の 2 つの方法によって行われる．

1. 対象のデバイスを占有する OS では，対象のデバイスにドライバを登録する．
2. 対象のデバイスを占有しない OS では，対象のデバイスにドライバを登録しない．

デバイスにドライバを登録した OS では，対象のデバイスを操作できるが，デバイスにドライバを登録していない OS では，対象のデバイスを操作できない．つまり，対象のデバイスにドライバを登録した OS で，対象のデバイスを占有できる．

3 バスとドライバ，およびデバイスの登録方法

OS がバスとドライバ，およびデバイスを登録する流れを図 1 に示し，以下で説明する．バスとドライバ，およびデバイスの登録処理は図 1 の「(1) カーネルのスタートアップルーチン」内の関数から一括して呼ばれる．バスとドライバ，およびデバイスの登録処理は，以下の 3 つの手順で登録を行う．

1. バスの登録を行う．

   図 1 の「(2)PCI バスの登録」において，OS にバスの情報を登録する．OS に PCI バス登録すると，システムに何も登録していない状態の図 2 から，図 3 のようになり，OS に PCI の情報が登録される．

2. サブシステムの登録を行う．

   図 1 の「(3)SCSI の登録」において，システムにサブシステム (バスの配下にあるバス) の情報を登録する．OS に SCSI を登録すると，図 4 のようになり，OS に SCSI の情報が登録される．

3. ドライバを登録し，ドライバに対応したデバイスを登録する．

   図 1 の「(4)SCSI ドライバの登録」「(5)PCI ドライバの登録」において，デバイスやバスにドライバを登録する．バスドライバ登録処理では，ドライバを登録しようとするバスの配下にさらにバスが接続されていれば，配下のバスドライバ登録処理を呼び出すとい

う再帰的処理を行う．また，配下がデバイスであれば，このデバイスを登録してドライバの登録処理が完了となる．

配下がデバイスである場合と配下がデバイスである場合のドライバとデバイスの登録方法を以下で説明する．

(a) 配下がデバイスである場合

配下がデバイスである場合のドライバ登録処の様子を図 5 に載せ，以下で説明する．SCSI のように配下がデバイスである場合，まず，SCSI のドライバ登録処理を行う．次に，SCSI バスに接続しているすべてのデバイスごとに，登録しようとするドライバが対象のデバイスで使用できるか確認する．使用できるなら，このデバイスにドライバを登録する．デバイスにドライバを登録すると OS から操作が可能となる．

(b) 配下がバスである場合

配下がデバイスである場合のドライバ登録処の様子を図 6 に載せ，以下で説明する．PCI のように配下がバスである場合，配下がデバイスである場合と同様に，PCI のドライバ登録処理を行う．PCI バスの配下には SCSI が接続されているので，再帰的に SCSI のドライバ登録処理を実行する．

4 バスとドライバ，およびデバイスの登録方法の詳細な処理の流れ

3 章で説明したバスとドライバ，およびデバイスの登録方法をソースコードの流れの図を用いて，詳細に説明する．

4.1 (1) カーネルのスタートアップルーチンの説明

図 1 の「(1) カーネルのスタートアップルーチン」のソースコードの流れを図 7 に示し，以下で説明する．バスの登録やドライバの登録はスタートアップルーチン内の do_one_initcall() 関数から一括して呼ばれる．(2)，(3) でバスやサブシステムをシステムに登録を行う．

(4)，(5) のドライバの登録では，それぞれのドライバを登録するために，module_init() 関数を呼び出す．module_init() 関数は，ドライバの初期設定を行い,「(A) ドライバ登録処理」に移る．

4.2 (A) ドライバを登録処理の説明

「(A) ドライバ登録処理」の流れを図 8 に示し，以下で説明する．

module_init() 関数から呼ばれた driver_register() 関数はバスにドライバを登録する処理に移る．

バスにドライバを登録する処理では，バスに登録されているすべてのデバイスごとにドライバのアタッチを行う __driver_attach() 関数を呼び出す．__driver_attach() 関数は，登録しようとするドライバがこのデバイスで使用できるか確認する．使用できるなら，driver_probe_device() 関数は probe 関数を呼び出す really_probe() 関数を呼び出す．really_probe() 関数では，まず，デバイスにドライバを登録する．次に，バスに probe 関数が登録されていれば，登録されている

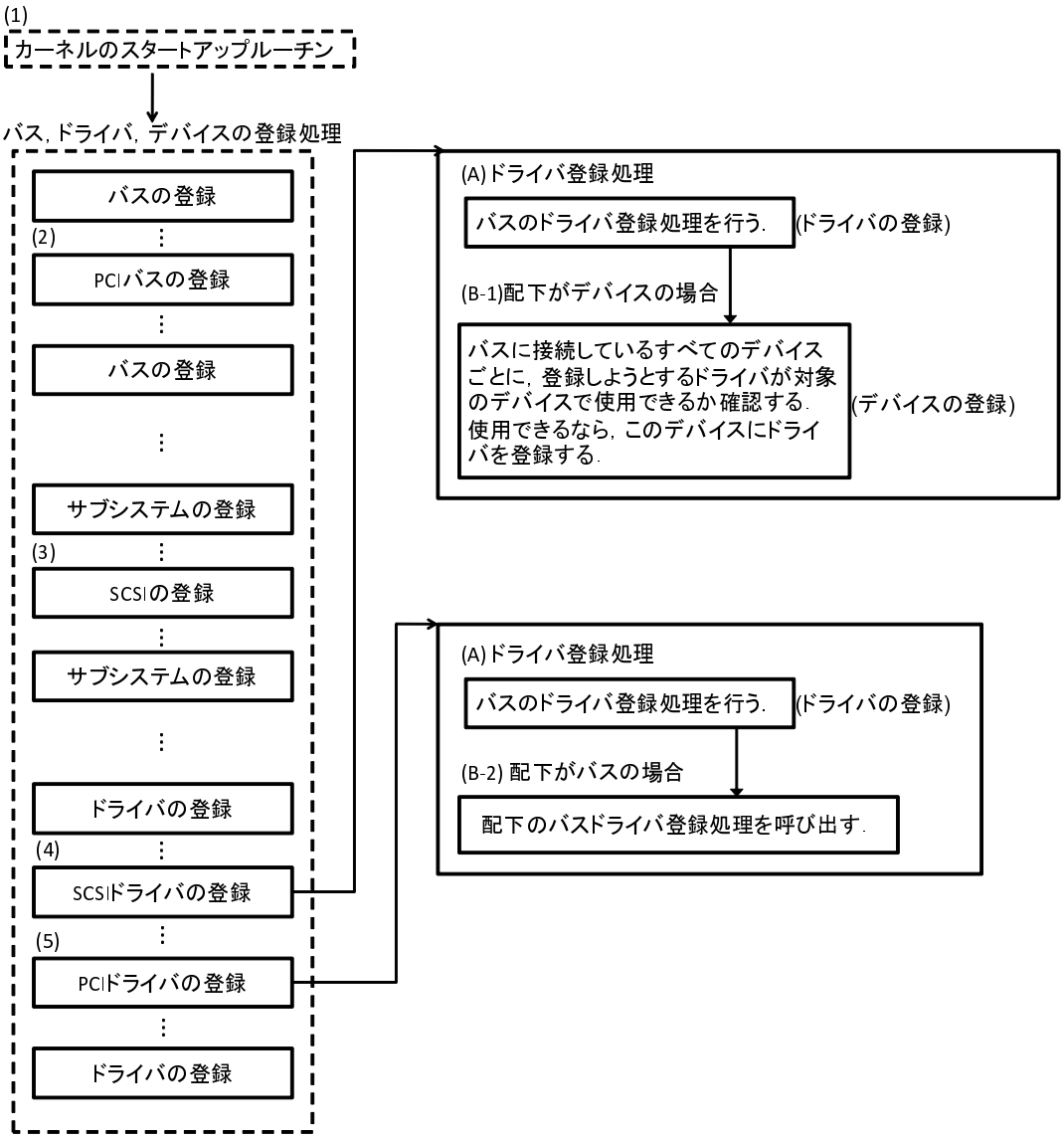

図 1: バス, ドライバ, デバイス登録流れ

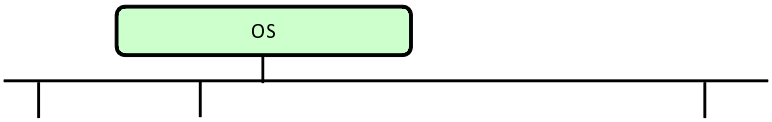

図 2: 初期

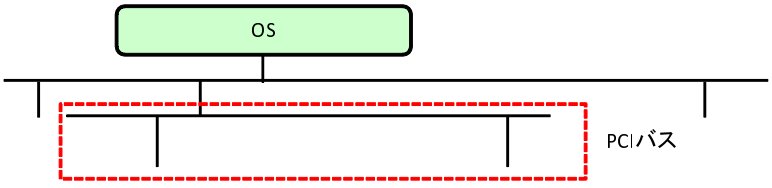

図 3: PCI の登録

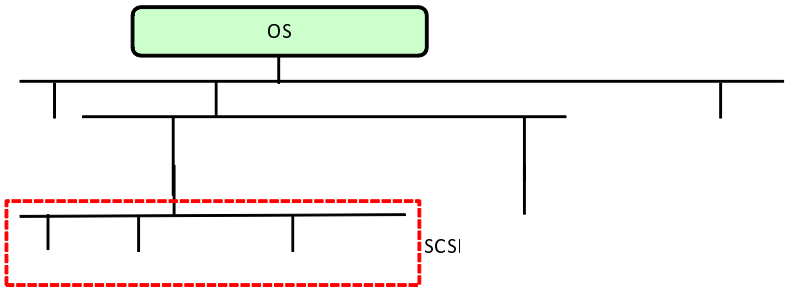

図 4: SCSI の登録

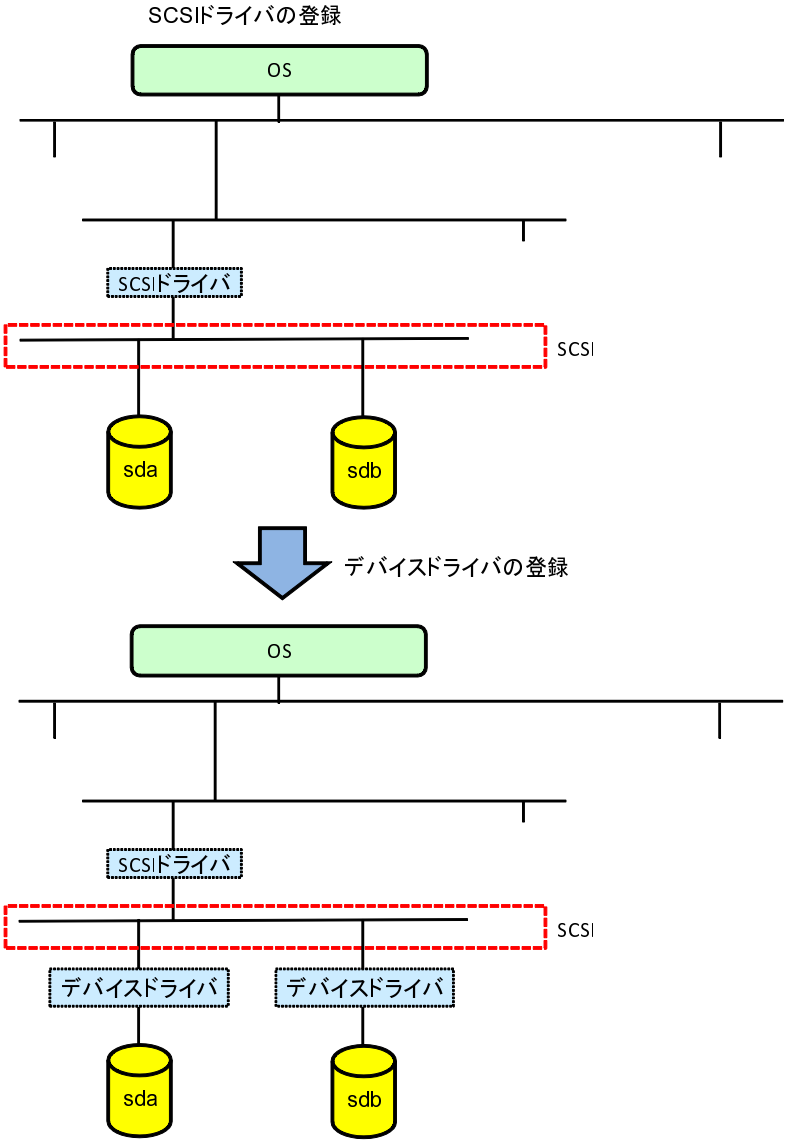

図 5: SCSI ドライバと対応するデバイスを登録

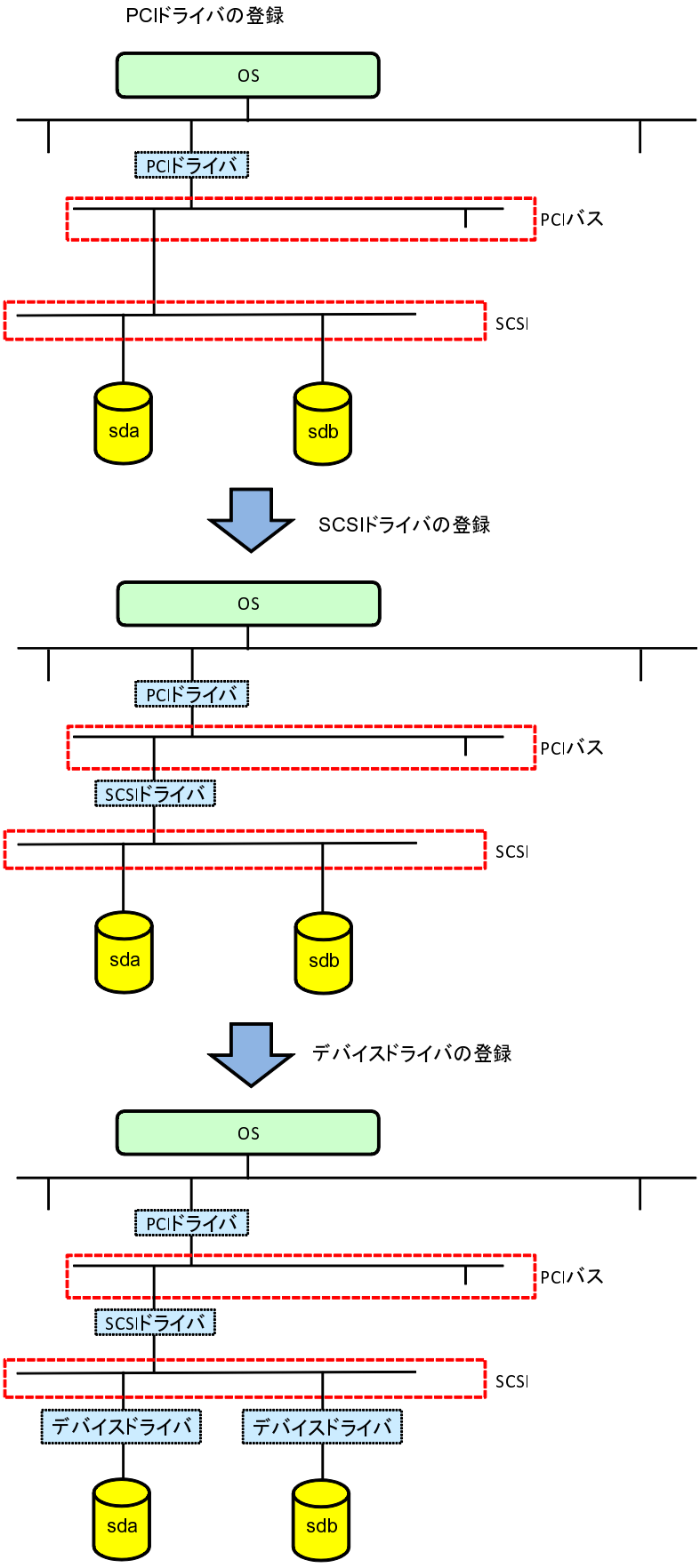

図 6: PCI ドライバと対応するデバイスを登録

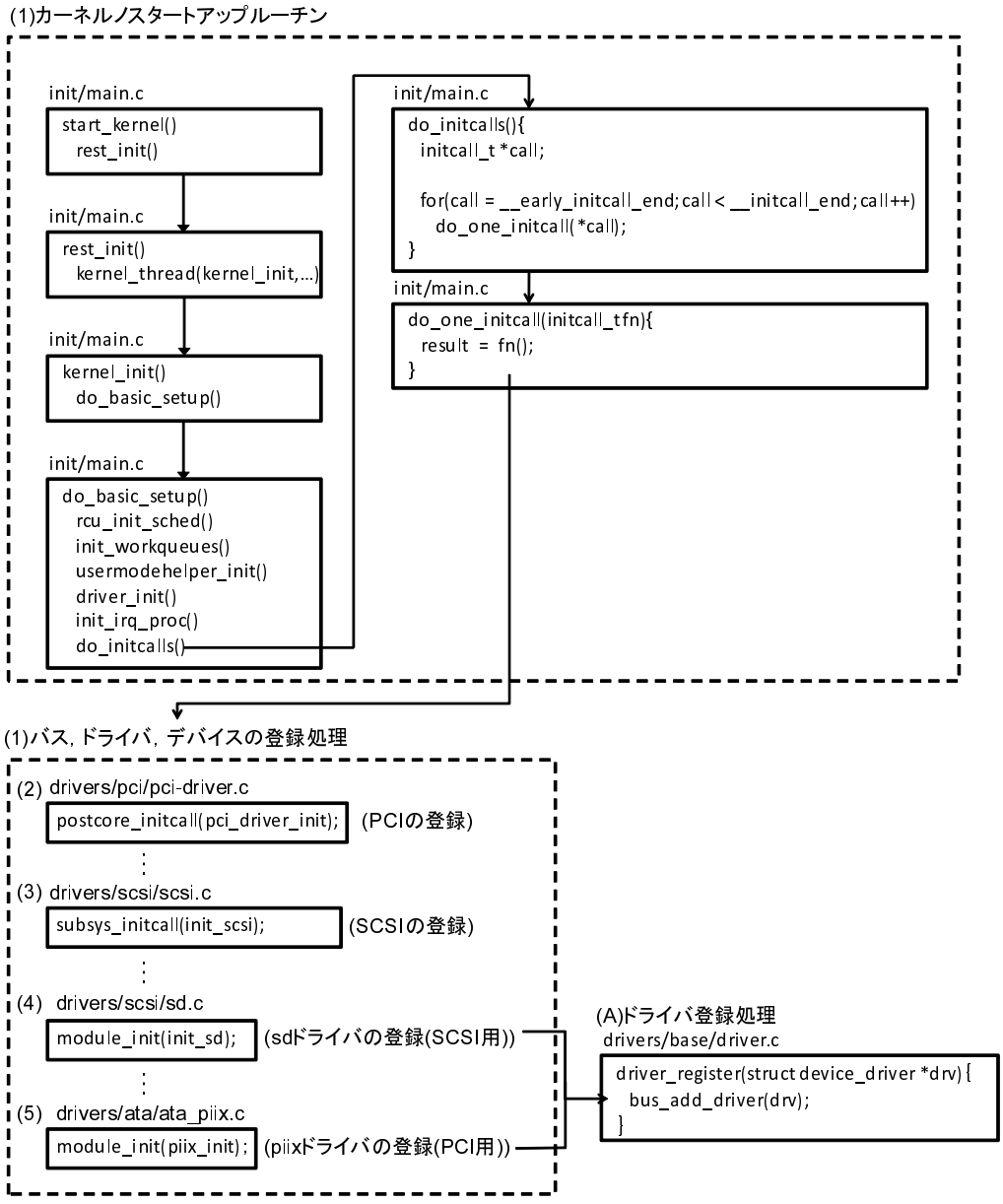

図 7: スタートアップルーチンからの呼び出し

probe 関数を実行する．バスに probe 関数が登録されていなければドライバの probe 関数を実行する．これらの probe 関数では，それぞれのドライバごとの処理を行い，配下がデバイスであれば，(B-1) のデバイスの attach 処理を行う．

配下がバスであれば，さらにこのバスにつながるデバイスを検索し，デバイスを登録する処理 (B-2) に移る．

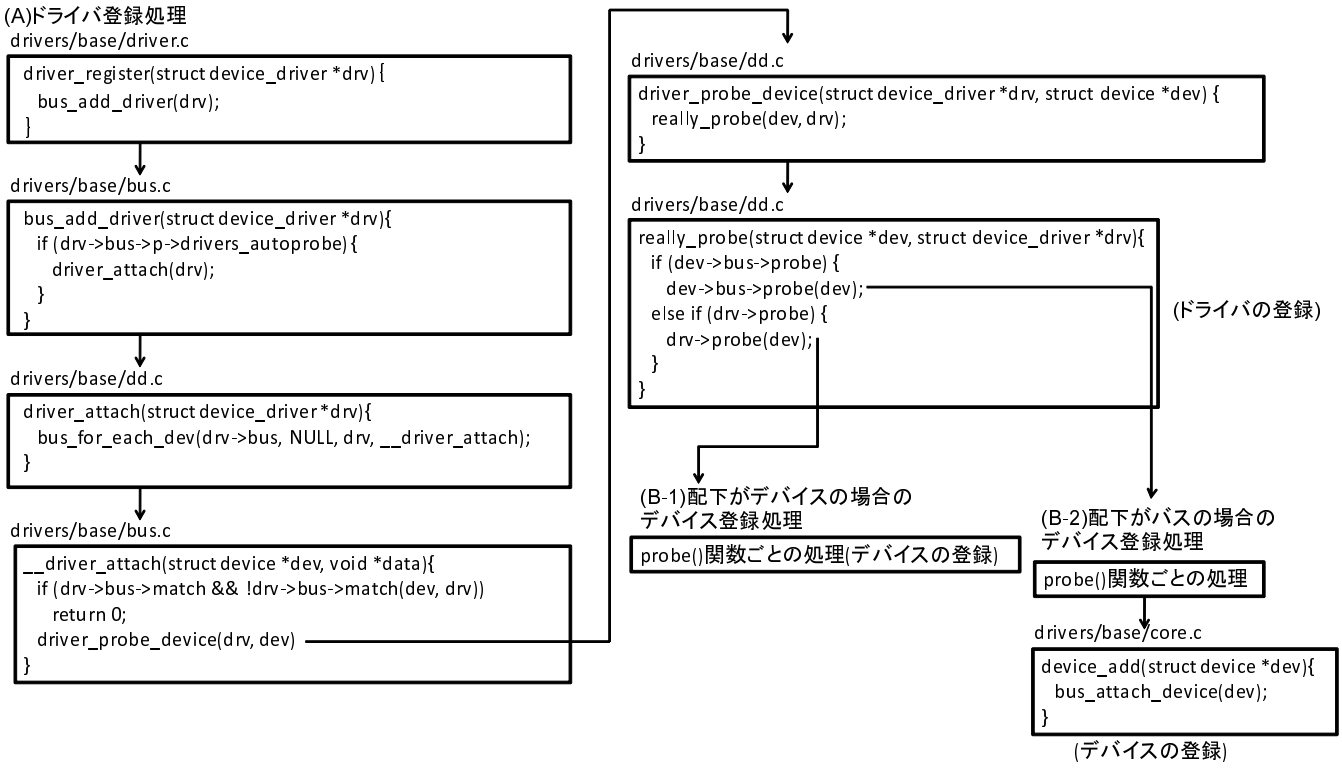

図 8: ドライバの登録

4.3 (B-2) デバイスを登録する処理の説明

「(B-2) デバイスを登録する処理」の流れを図 9 に示し，以下で説明する．

「(A) ドライバ登録処理」から呼ばれた device_add() 関数は，bus_attach_device() 関数を呼びバスにデバイスを登録する処理に移る．bus_attach_device() 関数では，バスに登録されているすべてのドライバごとにデバイスのアタッチを行う __device_attach() 関数を呼び出す．__device_attach() 関数は driver_probe_device() 関数を呼び出す．driver_probe_device() 関数は probe 関数を呼び出す really_probe() 関数を呼び出す．really_probe() 関数では，バスに登録されている probe 関数を実行する．バスに probe 関数が登録されていなければドライバの probe 関数を実行する．その後，probe() 関数ごとの処理 (デバイスの登録) を行う．

5 デバイスの判別

すべてのドライバは 3 章で説明した「(A) ドライバ登録処理」で登録される．手を加えていない Linux2.6.28.8 カーネルで，図 1 の「(A) ドライバ登録処理」において登録されるドライバ

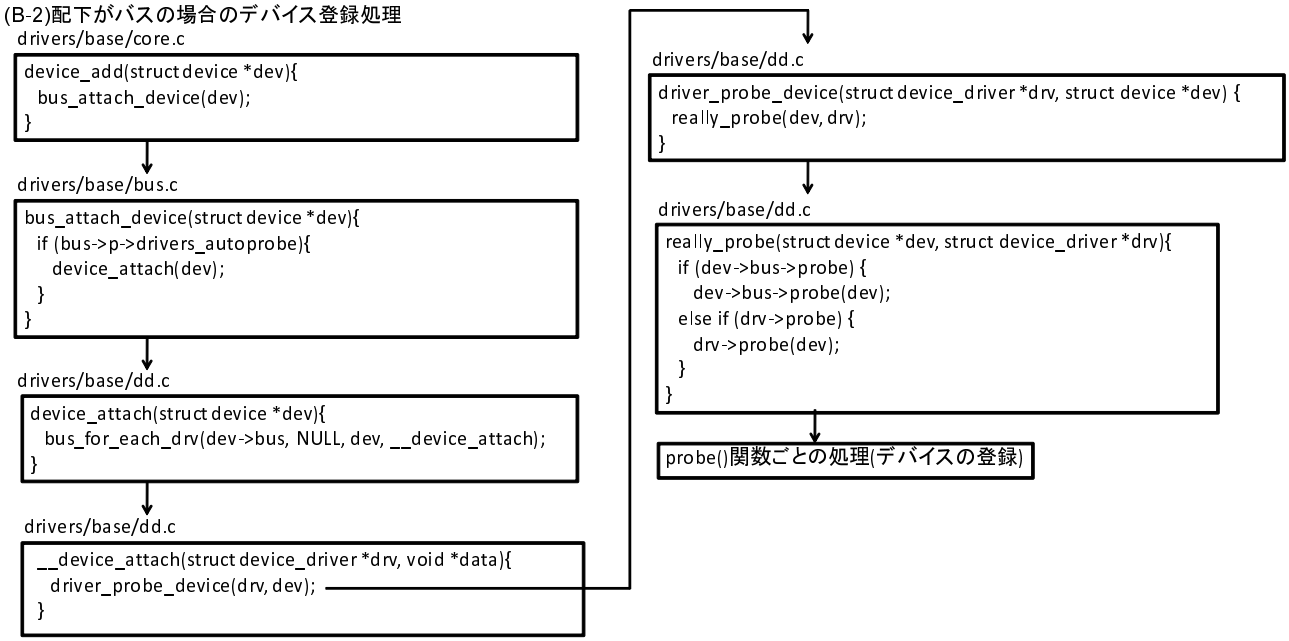

図 9: デバイスの登録

をリストアップした．具体的には，図 8 中の drivers/base/dd.c の driver_probe_device() 関数内で printk 関数を用いてリストアップした．さらに，登録する必要があるドライバと登録する必要がないドライバとを判断した．判断には，デバイスの情報が持っている bus_id(バス番号とデバイス番号を組み合わせたもの) を使用した．先行 OS と共存 OS で必要，不必要なものを判断した結果を，以下の表 1〜7 に載せる．

○: OS の起動に必須なデバイス
△: このデバイスの用途は不明だが，ドライバを登録しなくても OS の起動には影響しないデバイス
×: 不必要なデバイス

表 1: acpi バス

先行 OS	共存 OS	bus 名	bus_id	ドライバ名	備考
○	○	acpi	PNP0A03:00	pci_root	PCI のデバイス情報の取得
△	△	acpi	PNP0C0F:00	pci_link	用途不明
△	△	acpi	PNP0C0F:01	pci_link	用途不明
△	△	acpi	PNP0C0F:02	pci_link	用途不明
△	△	acpi	PNP0C0F:03	pci_link	用途不明
△	△	acpi	PNP0C0F:04	pci_link	用途不明
△	△	acpi	PNP0C0F:05	pci_link	用途不明
△	△	acpi	PNP0C0F:06	pci_link	用途不明
△	△	acpi	PNP0C0F:07	pci_link	用途不明
△	△	acpi	LNXPWRBN:00	button	用途不明
△	△	acpi	PNP0C0C:00	button	用途不明
△	△	acpi	ACPI_CPU:00	processor	用途不明
△	△	acpi	ACPI_CPU:01	processor	用途不明
△	△	acpi	LNXTHERM:00	thermal	用途不明

表 2: pnp バス

先行 OS	共存 OS	bus 名	bus_id	ドライバ名	備考
△	△	pnp	00:01	system	用途不明
△	△	pnp	00:0b	system	用途不明
△	△	pnp	00:0c	system	用途不明
△	△	pnp	00:0d	system	用途不明
×	○	pnp	00:07	serial	シリアルポート
×	○	pnp	00:08	serial	シリアルポート
○	×	pnp	00:0a	i8042 kbd	キーボード
△	△	pnp	00:03	rtc_cmos	用途不明

表 3: pci バス

先行 OS	共存 OS	bus 名	bus_id	ドライバ名	備考
△	△	pci	0000:00:00.0	agpgart-intel	Host bridge
×	×	pci	0000:00:1b.0	HDA Intel	Audio device のため不必要
△	△	pci	0000:00:1c.0	pcieport-driver	PCI bridge
△	△	pci	0000:00:1c.2	pcieport-driver	PCI bridge
×	×	pci	0000:00:1d.0	uhci_hcd	USB のため不必要
×	×	pci	0000:00:1d.1	uhci_hcd	USB のため不必要
×	×	pci	0000:00:1d.2	uhci_hcd	USB のため不必要
×	×	pci	0000:00:1d.3	uhci_hcd	USB のため不必要
×	×	pci	0000:00:1d.7	ehci_hcd	USB のため不必要
○	○	pci	0000:00:1f.1	ata_piix	IDE interface
○	○	pci	0000:00:1f.2	ata_piix	IDE interface
△	△	pci	0000:00:1f.3	i801_smbus	SMBus
×	×	pci	0000:02:00.0	tg3	Ethernet のため不必要

表 4: platform バス

先行 OS	共存 OS	bus 名	bus_id	ドライバ名	備考
×	○	platform	serial8250	serial8250	シリアルポート
○	×	platform	i8042	i8042	キーボード

表 5: scsi バス

先行 OS	共存 OS	bus 名	bus_id	ドライバ名	備考
×	×	scsi	0:0:0:0	sr	CD-DVD drive のため不必要
○	×	scsi	2:0:0:0	sd	sda のため先行 OS に必要
×	○	scsi	3:0:0:0	sd	sdb のため共存 OS に必要

表 6: usb バス

先行 OS	共存 OS	bus 名	bus_id	ドライバ名	備考
×	×	usb	usb1	usb	USB のため不必要
×	×	usb	usb2	usb	USB のため不必要
×	×	usb	usb3	usb	USB のため不必要
×	×	usb	usb4	usb	USB のため不必要
×	×	usb	usb5	usb	USB のため不必要
×	×	usb	1-0:1.0	hub	USB のため不必要
×	×	usb	2-0:1.0	hub	USB のため不必要
×	×	usb	3-0:1.0	hub	USB のため不必要
×	×	usb	4-0:1.0	hub	USB のため不必要
×	×	usb	5-0:1.0	hub	USB のため不必要
×	×	usb	4-2	usb	USB のため不必要
×	×	usb	4-2:1.0	usbhid	USB のため不必要

表 7: serio，hid バス

先行 OS	共存 OS	bus 名	bus_id	ドライバ名	備考
○	×	serio	serio0	atkbd	キーボード
△	△	hid	0003:0461:4D15.0001	generic-usb	用途不明

6 先行 OS における必要なデバイスの登録

先行 OS において，5 章で○をつけたデバイスのみ認識させる．そこで，drivers/base/dd.c の driver_probe_device() 関数において対象のデバイスのみ really_probe() 関数を実行させ，ドライバの登録処理を行うように改造した．改造を加えたソースコード以下に示し説明する．なお，追加された行を+で，削除された行を-で示す．

```
drivers/base/dd.c
     1   int driver_probe_device(struct device_driver *drv, struct device *dev)
     2   {
     3      int ret = 0;
     4      if (!device_is_registered(dev))
     5      return -ENODEV;
     6      if (drv->bus->match && !drv->bus->match(dev, drv))
     7      goto done;
     8  -   ret = really_probe(dev, drv);
     9  -
    10  +/* Relation:TwinOS, Author:senzaki, Date:2009.10.26
    11  + * Note: PreOS occupy the device.
    12  + * PNP0A03      "get PCI device information"
    13  + * 00:0a        "keyboard"
    14  + * 0000:00:1f.1 "IDE interface"
    15  + * 0000:00:1f.2 "IDE interface"
    16  + * i8042        "keyboard"
    17  + * 2:0:0:0      "sda"
    18  + * serio0       "keyboard"
    19  + */
    20  +   if(
    21  +   !strcmp(dev->bus_id,"PNP0A03:00") ||
    22  +   !strcmp(dev->bus_id,"00:0a") ||
    23  +   !strcmp(dev->bus_id,"0000:00:1f.1") ||
    24  +   !strcmp(dev->bus_id,"0000:00:1f.2") ||
    25  +   !strcmp(dev->bus_id,"i8042") ||
    26  +   !strcmp(dev->bus_id,"2:0:0:0") ||
    27  +   !strcmp(dev->bus_id,"serio0")
    28  +   ){
    29  +       ret = really_probe(dev, drv);
    30  +   }
    31   done:
    32      return ret;
    33   }
```

bus_id はデバイス構造体のメンバであり char 型の配列として宣言されている．そこで，strcmp 関数 (lib/string.c 247 行目で定義 第 1 引き数と第 2 引数の文字列を比較し一致なら 0，不一致なら 1 を返す) を用いて 4 章で判断した必要なデバイスの bus_id と比較し，一致したときのみ really_probe() 関数を実行する．

7 まとめ

NewOS におけるデバイスの占有方法について説明した．4 章の△で示した必要か不必要か判断できないものについては別途調査が必要である．また，実験用マシンが変わると認識するデバイスも変わるので，注意が必要である．

本資料で説明した実装によるカーネルへの変更は TwinOS2.6 用ギットリポジトリ [1] から，以下のタグ，もしくはコミット番号で参照できる．

- pre-occupy_device
- 14fd15fadb09e0d4c56725f75874309f76c49ff9

参考文献

[1] ”TwinOS2.6 用 Git リポジトリ (ブラウジング用)，”http://newgroup.swlab.cs.okayama-u.ac.jp/gitweb/cgi-bin/gitweb.cgi?p=TwinOS26.git;a=summary