Nouveautés du noyau Linux 2.6

Modèle unifié de périphériques

Stelian Pop <stelian.pop@fr.alcove.com>

Nouveautés du noyau Linux 2.6

Cycle de développement

- ✓ cycle de développement **2.3** lancé le *11/05/1999*
- ✓ noyau **2.4.0** publié le *04/01/2001*
- ✓ noyau **2.4.24** publié le *05/01/2004*

- ✓ cycle de développement **2.5** lancé le *22/11/2001*
- ✓ noyau **2.6.0-test1** publié le *13/07/2003*
- ✓ noyau **2.6.0** publié le *17/12/2003*

Retour

Fermer

Taille du noyau

Noyau **2.4.24** :

✓ taille de l'archive `tar.bz2` : *28M*

✓ taille totale du code : *3480 KSLOC*[1]

– drivers : *1905 KSLOC*

– core : *1575 KSLOC*

Noyau **2.6.1** :

✓ taille totale de l'archive `tar.bz2` : *32M*

✓ taille totale du code : *3871 KSLOC*

– drivers : *1835 KSLOC*

– core : *2036 KSLOC*

[1]Comptage effectué grâce à SLOCCount, http://www.dwheeler.com/sloccount/

Retour

Fermer

Mainteneurs

Noyaux **2.4** :
Marcelo Tosatti <marcelo.tosatti@cyclades.com>

Noyaux **2.6** :
Andrew Morton <akpm@osdl.org>

Noyaux **2.7** :
Linus Torvalds <torvalds@osdl.org>

Support des architectures

- ✓ Nouvelles architectures supportées : *x86-64* (*AMD Hammer*), *ppc64*, *uClinux* (*Motorola m68knommu : Dragonball, Coldfire*), *Hitachi h8300*, *NEC v850*, *UML*
- ✓ Compilation optimisée possible pour l'embarqué
- ✓ Support des architectures *NUMA*
- ✓ Découpage des architectures en subarchitectures
- ✓ Support de l'*hyperthreading* et *PAE 64 GB Intel*

Fonctionnalités internes

- ✓ Support de la *préemptivité* en mode noyau
- ✓ Support d'un grand nombre d'utilisateurs/groupes (*uid/gid 32 bit*)
- ✓ Support d'un grand nombre de processus (*pid 32 bit*)
- ✓ *Ordonnanceur O(1)* optimisé pour un grand nombre de processus
- ✓ *Futexes* : sémaphores optimisées pour les processus
- ✓ *NPTL* : bibliothèque optimisée de gestion des threads POSIX
- ✓ Extension du nommage des périphériques (passage du *major/minor* de 8/8 bit à *12/20 bit*)
- ✓ Réécriture de la *gestion des modules* (optimisation du chargement, élimination des problèmes lors du déchargement, gestion des arguments)
- ✓ Amélioration du support *ACPI*
- ✓ Nouveau *modèle de périphériques* (organisation hiérarchique et transparente de tous les périphériques)
- ✓ Implémentation (initiale) de l'*hibernation*




Retour
Fermer

Périphériques en mode bloc

- ✓ Optimisation des algorithmes d'*ordonnancement des entrées/sorties*
- ✓ Implémentation d'*AIO* au niveau noyau
- ✓ *Capacité* des périphériques sur *64 bit* (15 TB sur architecture 32 bits)
- ✓ LVM1 supprimé du noyau et remplacé par *device mapper*

Systèmes de fichiers

- ✓ Optimisation des systèmes de fichiers *ext2/ext3* (*EA, ACL, répertoires indexés, allocateur Orlov*)
- ✓ Simplification de *devfs* (migration vers *udev*)
- ✓ Rajout de nouveaux systèmes de fichiers : *XFS, JFS, CIFS, AFS*
- ✓ Réécriture de implémentation des *quotas*

Réseau

- ✓ Nombreuses optimisations
- ✓ Rajout de *IPsec*
- ✓ Support de *NFSv4*
- ✓ Support de *NFS* par dessus *TCP*

Multimédia

- ✓ Support de la *préemptivité* au niveau du noyau
- ✓ Amélioration des temps de latence (*low-latency*)
- ✓ Amélioration de la couche *framebuffer* (rotation, accélération etc)
- ✓ Adoption de *ALSA* en tant que sous-système sonore, destiné à remplacer OSS
- ✓ Migration vers *Video4Linux v2*
- ✓ Support de périphériques *DVB*




Périphériques

- ✓ Support de la norme *USB 2.0*
- ✓ Support des périphériques *USB esclave* (*usb gadget*)
- ✓ Incorporation des *lm_sensors*
- ✓ Réécriture de la couche *IDE* (support du *Serial-ATA*, *TCQ*, *DMA* vers les graveurs etc...)
- ✓ Réécriture de la couche *SCSI*
- ✓ Réécriture de la couche d'interfaçage avec l'utilisateur (*HID*) : déconnexion entre la notion de console et notion de clavier/écran, nouveaux périphériques supportés etc.
- ✓ ...

Sécurité

- ✓ API générique permettant le *cryptage* : *HMAC*, *MD4*, *MD5*, *SHA-1*, *SHA256*, *SHA384*, *SHA512*, *DES*, *Triple DES EDE*, *Blowfish*, *Twofish*, *Serpent*, *AES*, *CAST5*, *CAST6*
- ✓ Rajout de la cryptographie dans le pilote *loop* (*cryptoloop*)
- ✓ Migration vers un système de droits gérés par *capabilités*, géré par une couche générique
- ✓ Incorporation de *SELinux* en tant qu'implémentation spécifique des capabilités

Déboggage

- ✓ *Configuration* et *compilation* améliorés du noyau (optimisation des dépendances, compilation partielle, syntaxe simplifiée, etc)
- ✓ Possibilité d'inclure le nom des symboles dans le binaire du noyau (afin d'avoir des oops décodés) : *kksymoops*
- ✓ Possibilité d'inclure les informations sur la configuration du noyau, la version de compilateur utilisée etc. dans le binaire du noyau : *kconfig*
- ✓ Inclusion de *Oprofile*

Modèle unifié de périphériques (device model)




Problèmatique

Dans le noyau **2.4**, chaque bus gère ses périphériques indépendemment (*PCI*, *USB*, etc) $\longrightarrow$ pas de relation d'ordre ou de dépendance entre les périphériques.

Conséquence : impossible d'avoir une gestion commune de tous les périphériques (ex : hibernation, accès aux périphériques unifié etc).

Solution

Le *modèle unifié de périphériques* développé par *Patrick Mochel* <mochel@osdl.org> apporte une réponse à cette problématique.

Il permet de :

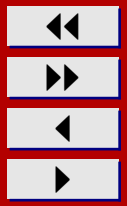
- ✓ connaître l'ensemble des périphériques de la machine et leur état
- ✓ connaître la structure des bus de la machine
- ✓ connaître l'ensemble des pilotes de périphériques de la machine
- ✓ présenter les périphériques à la fois par utilisation et par connexion
- ✓ exporter les paramètres des périphériques (remplacement de `ioctl()` et `/proc`)

Voir : `Documentation/kobject.txt`

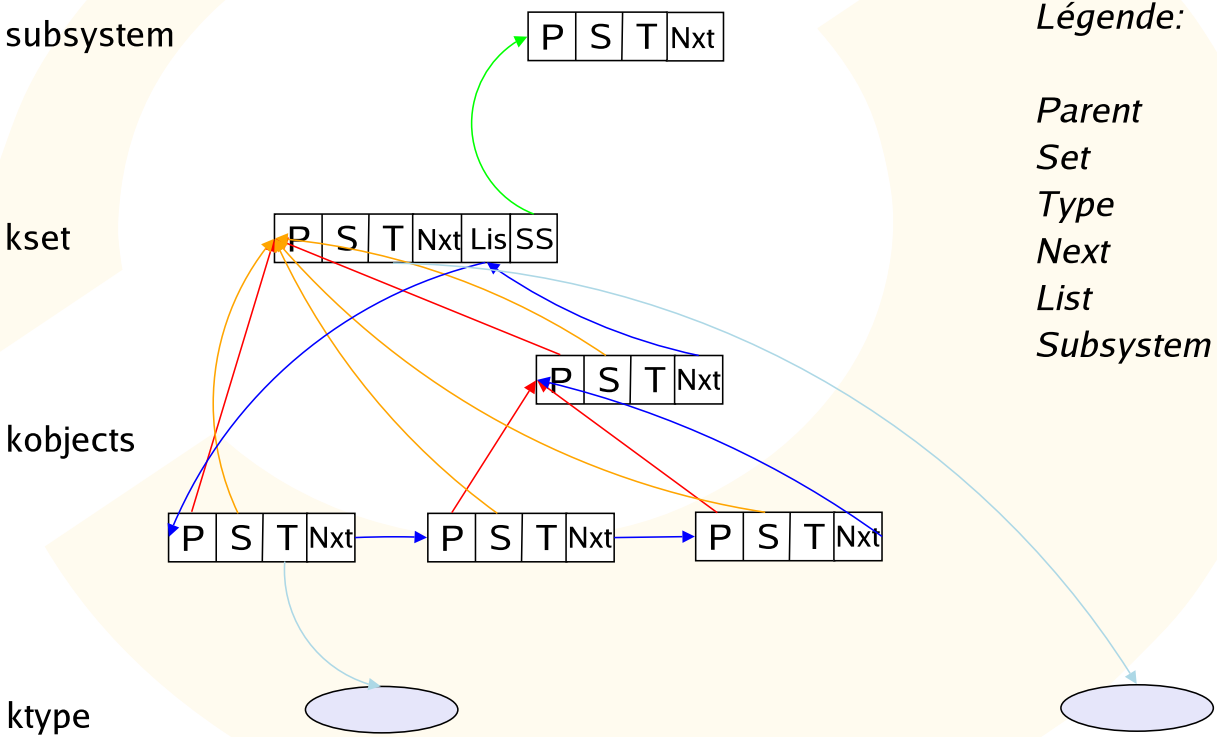
Structure du modèle unifié de périphériques

La structure du modèle unifié s'appuie sur des nouvelles structures :

- ✓ `kobject` : "classe mère" de tous les éléments du modèle, contenu dans tous les objets, qui contient (des liens vers) : nom, compteur de référence, parent, `kset`, `kobj_type`
- ✓ `kobj_type` : type commun de plusieurs `kobject`, implémente la libération des `kobject`, la façon de représenter un `kobject` dans *sysfs*
- ✓ `kset` : liste de `kobject`, généralement partageant un même `kobj_type`, peut contenir d'autre `kset`, attaché à un `subsystem`
- ✓ `subsystem` : entité de base du système, contient des `kset` (vue du système).

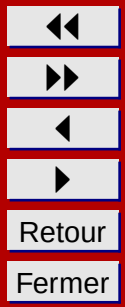


Structure du modèle unifié... - suite



Légende:

Parent
Set
Type
Next
List
Subsystem



sysfs

Le modèle unifié de périphérique exporte son état sous la forme d'un système de fichiers virtuel.

sysfs est destiné à être monté sur `/sys`

Chaque `kobject` donne lieu à un répertoire dans *sysfs*. Des liens symboliques sont utilisées pour implémenter les différentes vues.

L'accès aux attributs des bus/périphériques/pilotes est simplifié : un fichier texte par paramètre.

Problématique supplémentaire : libération détachée des ressources (car référence possible sur les entrées dans *sysfs*).

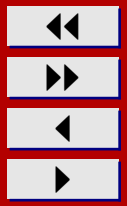
Voir : `Documentation/filesystems/sysfs.txt`

Structure de *sysfs*

`/sys` contient tous les *subsystems* :

- ✓ `block` : liste des pilotes en mode bloc
- ✓ `bus` : liste des bus
- ✓ `cdev` : liste des pilotes en mode caractère
- ✓ `class` : liste des périphériques par utilisation
- ✓ `devices` : liste des périphériques par connexion
- ✓ `firmware` : liste des périphériques par firmware (ACPI)

Chacun de ces *subsystems* dispose de sa propre API d'ajout / suppression / définition d'élements, voir `Documentation/driver-model/*`



Structure de *sysfs* - suite

```
/sys
|-- block
|   |-- hda
|   |   |-- dev
|   |   |-- device -> ../../devices/pci0000:00/0000:00:07.1/ide0/0.0
|   |   |-- hda1
|   |   |   |-- dev
|   |   |   |-- size
|   |   |   |-- start
|   |   |   `-- stat
...
|-- bus
|   |-- ide
|   |   |-- devices
|   |   |   `-- 0.0 -> ../../../devices/pci0000:00/0000:00:07.1/ide0/0.0
|   |   `-- drivers
|   |       `-- ide-disk
...
|-- devices
|   |-- pci0000:00
|   |   |-- 0000:00:07.1
|   |   |   |-- class
|   |   |   |-- config
|   |   |   |-- detach_state
|   |   |   |-- device
|   |   |   |-- ide0
|   |   |   |   |-- 0.0
|   |   |   |   |   |-- block -> ../../../../../block/hda
|   |   |   |   |   |-- detach_state
|   |   |   |   |   `-- power
|   |   |   |   |       `-- state
|   |   |   |   |-- detach_state
|   |   |   |   `-- power
|   |   |   |       `-- state
|   |   |   |-- irq
|   |   |   |-- power
|   |   |   |   `-- state
|   |   |   |-- resource
|   |   |   |-- subsystem_device
|   |   |   |-- subsystem_vendor
|   |   |   `-- vendor
```

Retour

Fermer

Exemple d'attributs d'un périphérique

```
static int foo_device;

static ssize_t foo_device_show(struct device *dev,
       char *buf)
{
        return snprintf(buf, PAGE_SIZE, "%d\n", foo_device);
}

static ssize_t foo_device_set(struct device *dev,
      const char *buf, size_t count)
{
        sscanf(buf, "%d", &foo_device);
        return count;
}

DEVICE_ATTR(foo_device, 0644, foo_device_show, foo_device_set);

...

device_create_file(&meye.mchip_dev->dev, &dev_attr_foo_device);

...

device_remove_file(&meye.mchip_dev->dev, &dev_attr_foo_device);
```

Exemple d'attributs d'un périphérique - suite


```
# mount | grep sys
none on /sys type sysfs (rw)
# ls /sys
block  bus  cdev  class  devices  firmware  power
# find /sys -name "*foo*"
/sys/devices/pci0000:00/0000:00:0b.0/foo_device
# ls /sys/devices/pci0000:00/0000:00:0b.0/
class   detach_state  foo_device  power     subsystem_device  vendor
config  device        irq         resource  subsystem_vendor
# cat /sys/devices/pci0000:00/0000:00:0b.0/foo_device
0
# echo "42" > /sys/devices/pci0000:00/0000:00:0b.0/foo_device
# cat /sys/devices/pci0000:00/0000:00:0b.0/foo_device
42
```

Exemple d'attributs d'un pilote

```
static int bar_driver;

static ssize_t bar_driver_show(struct device_driver *dd,
       char *buf)
{
        return snprintf(buf, PAGE_SIZE, "%d\n", bar_driver);
}

static ssize_t bar_driver_set(struct device_driver *dd,
      const char *buf, size_t count)
{
        sscanf(buf, "%d", &bar_driver);
        return count;
}

DRIVER_ATTR(bar_driver, 0644, bar_driver_show, bar_driver_set);

...

driver_create_file(&meye_driver.driver, &driver_attr_bar_driver);

...

driver_remove_file(&meye_driver.driver, &driver_attr_bar_driver);
```

Exemple d’attributs d’un pilote - suite

```
# mount | grep sys
none on /sys type sysfs (rw)
# ls /sys
block  bus  cdev  class  devices  firmware  power
# find /sys -name "*bar*"
/sys/bus/pci/drivers/meye/bar_driver
# ls /sys/bus/pci/drivers/meye/
0000:00:0b.0  bar_driver  new_id
# cat /sys/bus/pci/drivers/meye/bar_driver
0
# echo "42" > /sys/bus/pci/drivers/meye/bar_driver
# cat /sys/bus/pci/drivers/meye/bar_driver
42
```

Futur du *modèle unifié de périphériques*

- ✓ déprécation d'une partie de `/proc`
- ✓ remplacement de *devfs* par *udev*
- ✓ interface standard de chargement de firmware par *sysfs*
- ✓ possibilité d'implémenter l'hibernation
- ✓ ...

Conclusion

Le noyau **2.6** apporte plein de nouvelles fonctionnalités et améliorations, reste à attendre son support par les distributions Linux.

De nombreuses bases ont été posés dans le noyau **2.6** qui restent à être exploités dans le futur...

Liens

- Wonderful World of Linux 2.6
  http://kniggit.net/wwol26.html
- Dave Jones' Post-halloween doc
  http://www.codemonkey.org.uk/docs/post-halloween-2.6.txt
- Articles LWN sur l'API du noyau 2.6
  http://lwn.net/Articles/driver-porting/
- Alcôve
  http://www.alcove.com
- Stelian Pop
  http://popies.net
  <stelian.pop@fr.alcove.com>
  <stelian@popies.net>