UNIVERSITÉ
CÔTE D'AZUR

Modules

Présentation: Stéphane Lavirotte
Auteurs: ... et al*

(*) Cours réalisé grâce aux documents de :
Olivier Dalle, Erick Gallesio, Fabrice Huet, Stéphane Lavirotte,
Michael Opdenacker, Jean-Paul Rigault

Mail: Stephane.Lavirotte@unice.fr
Web: http://stephane.lavirotte.com/
Université Côte d'Azur

UNIVERSITÉ
CÔTE D'AZUR

Introduction

Pour un bon départ

UNIVERSITÉ CÔTE D'AZUR

D'où viennent les Modules

- ✓ Principe inspiré des micro-noyaux (ex: Mach) :
  - Le (micro)noyau ne contient que du code générique
    - synchronisation
    - ordonnancement
    - Inter-Process Communications (IPC) ...
  - Chaque couche/fonction/pilote de l'OS est écrit(e) de façon indépendante
    - Obligation de définir et utiliser des interfaces/protocoles clairement définis
    - Fort cloisonnement (module MACH = processus)
    - Meilleure gestion des ressources
      - Seuls les modules actifs consomment des ressources

UNIVERSITÉ CÔTE D'AZUR

Les Modules vu par Linux

- ✓ Module = fichier objet
  - Implémentation très partielle du concept de micro-noyau
  - Code pouvant être lié dynamiquement à l'exécution
    - Lors du boot (`rc` scripts)
    - A la demande (noyau configuré avec option CONFIG_KMOD)
  - Mais le code peut aussi, généralement, être lié statiquement au code du noyau (approche monolithique traditionnelle)
- ✓ Avantages
  - Ajout de fonctionnalités au noyau (pilotes, support FS, ...)
  - Développer des pilotes sans redémarrer: chargement, test, déchargement, recompilation, chargement...
  - Supporter l'incompatibilité entre pilotes
  - Utile pour garder une image du noyau à une taille minimum
  - Une fois chargé, accès à tout le noyau. Aucune protection particulière.

UNIVERSITÉ
CÔTE D'AZUR

Liaisons Dynamiques 1/3

- ✓ Le code d'une librairie n'est pas copié dans l'exécutable à la compilation
  - Il reste dans un fichier séparé (fichier .ko depuis Linux 2.6)
- ✓ Le linker ne fait pratiquement rien à la compilation
  - Il note les librairies dont un exécutable à besoin (voir Dépendances des Modules)
- ✓ Le gros du travail est fait au chargement ou à l'exécution
  - Par le loader de l'OS
- ✓ Le loader cherche/charge les bibliothèques dont un programme a besoin
  - Ajoute les éléments nécessaires à l'espace d'adressage du processus

UNIVERSITÉ CÔTE D'AZUR

Liaisons Dynamiques 2/3

- ✓ Utiliser la liaison dynamique implique des relocations
  - Les adresses des sauts ne sont pas connues à la compilation
  - Elles ne sont connues que lorsque le programme et les bibliothèques sont chargées
  - Impossible de pré-allouer des espaces
    - Conflits
    - Limitations de l'espace mémoire en 32 bits
- ✓ Utilisation d'une table d'indirection
  - Une table vide est ajoutée au programme à la compilation
  - Toutes les références passent par cette table (import directory)
  - Les bibliothèques ont une table similaire pour les symboles qu'elles exportent (entry points)
  - Cette table est remplie au chargement par le loader
- ✓ Plus lent que de la liaison statique

UNIVERSITÉ CÔTE D'AZUR

Liaisons Dynamiques 3/3

- ✓ Comment trouver la bibliothèque à l'exécution ?

- ✓ Unix
  - Répertoires bien connus
  - Spécifiés dans `/etc/ld.so.conf`
  - Variable d'environnement (`LD_PRELOAD`, `LD_LIBRARY_PATH`)

- ✓ Windows
  - Utilisation du registry pour ActiveX
  - Répertoires bien connus (System32, System...)
  - Ajout de répertoires par SetDllDirectory()
  - Répertoire courant et PATH

UNIVERSITÉ CÔTE D'AZUR

Module Nécessaire au Démarrage

- ✓ Compilation d'un noyau
  - Si « tout » est compilé en module :
    - Comment démarrer le système ?
  - Exemple:
    - Partition contenant le système de fichier racine en ext3
    - ext3 n'est pas compilé statiquement dans le noyau
    - Comment accéder au système de fichier racine ?

- ✓ La solution: `initrd`

UNIVERSITÉ CÔTE D'AZUR

`initrd`: Initial Ram Disk

- ✓ Initrd (Initial RAM disk): disque mémoire de démarrage
  - Système de fichier racine (/) minimaliste en RAM
  - Utilisé traditionnellement pour minimiser le nombre de pilotes de périphériques compilés statiquement dans le noyau.
  - Utile aussi pour lancer des scripts d'initialisation complexes
  - Utile pour charger des modules propriétaires (qui ne peuvent être liés statiquement au noyau)

- ✓ Pour en savoir plus:
  - Lire `Documentation/initrd.txt` dans les sources du noyau !
  - Couvre aussi le changement de système de fichier racine («pivot_root»)

UNIVERSITÉ CÔTE D'AZUR

Créer Manuellement une Image initrd

```
mkdir /mnt/initrd

dd if=/dev/zero of=initrd.img bs=1k count=2048

mkfs.ext2 -F initrd.img

mount -o loop initrd.img /mnt/initrd

<Remplir avec: les modules, le script linuxrc…>

umount /mnt/initrd

gzip --best -c initrd.img > initrd
```

Ou

```
mkinitramfs ou mkinitrd
```

UNIVERSITÉ
CÔTE D'AZUR

Les Modules

Ajout de Fonctionnalités au Noyau

UNIVERSITÉ CÔTE D'AZUR

Modules Dynamiques – Principes

- ✓Le noyau permet de gérer un grand nombre de composants
- ✓Ceux-ci peuvent ne pas être actifs en même temps
- ✓Les modules permettent de charger les composants du noyau au moment où il sont nécessaires (et s'ils sont nécessaires)
- ✓Avantages:
  - Moins de mémoire consommée par le noyau donc plus de mémoire pour les utilisateurs
  - Eviter une compilation totale du noyau lors du rajout d'un périphérique
  - Facilite le debug d'un driver: on peut décharger puis recharger le module (si on a pas planté la machine ;-)

UNIVERSITÉ CÔTE D'AZUR

Module « Hello World »

✓ Module « Hello World »: hello.c

```
#include <linux/module.h>  /* Nécessaire pour tout module */

static int init_module(void)
{
        printk("Hello World!\n");
        return 0;
}

static void cleanup_module(void)
{
        printk("Goodbye, cruel world!\n");
}

MODULE_LICENSE("GPL");
```

UNIVERSITÉ CÔTE D'AZUR

Initialisation du Module

- ✓ **Depuis les noyaux 2.2:**
  - **Macros `__init` et `__exit`**
  - **`__init`: si « built-in », mémoire libérée après initialisation**

    `static int __init init_module(void)`
  - **`__exit`: si « build-in", omission de cette fonction**

    `static void __exit cleaup_module(void)`
- ✓ **Depuis les noyaux 2.4:**
  - **Possibilité de donner des noms différents à `init|clean_module()`**

```
#include <linux/init.h>  /* Nécessaire pour les macros */

  static int __init init_function (void) { … }

  static void __exit exit_function (void) { … }

  module_init(init_function);
  module_exit(exit_function);
```

UNIVERSITÉ CÔTE D'AZUR

Macros de Documentation

- ✓ **Documentation du module**
  - `MODULE_AUTHOR("...");`
  - `MODULE_DESCRIPTION("...");`
  - `MODULE_SUPPORTED_DEVICE("...");`
- ✓ **Depuis les noyaux 2.4:**
  - **Nécessité de définir une licence pour un module sinon:**

```
> insmod module.o

Warning: loading module.o will taint the kernel: no licence
  See http://www.tux.org/lkml/#export-tainted for information about
   tainted modules
On entre dans le module
Module module loaded, with warnings
```

UNIVERSITÉ CÔTE D'AZUR

Licence des Modules

`MODULE_LICENSE("...");` **voir** `include/linux/module.h`

- ✓ GPL
  - GNU Public License v2 ou supérieure
- ✓ GPL v2
  - GNU Public License v2
- ✓ GPL and additional rights
- ✓ Dual BSD/GPL
  - Choix entre GNU Public License v2 et BSD
- ✓ Dual MIT/GPL
  - Choix entre GNU Public License v2 et MIT
- ✓ Dual MPL/GPL
  - Choix entre GNU Public License v2 et Mozilla
- ✓ Propriétaire
  - Produits non libres

UNIVERSITÉ CÔTE D'AZUR

Contraintes de Licence sur Linux

- ✓ Aucune obligation de redistribuer
  - Vous pouvez partager vos modifications dès le début dans votre propre intérêt, mais n'y êtes pas obligés !
- ✓ Contraintes au moment de distribuer
  - Pour tout périphérique embarquant Linux et des Logiciels Libres, vous devez distribuer vos sources à l'utilisateur final. Vous n'avez aucune obligation de les distribuer à qui que se soit d'autre !
  - Les modules propriétaires sont tolérés (mais non recommandés) tant qu'ils ne sont pas considérés comme dérivés de code GPL.
  - Les pilotes propriétaires ne peuvent pas être liés statiquement au noyau.
  - Aucun soucis pour les pilotes disponibles sous une licence compatible avec la GPL (détails dans la partie sur l'écriture de modules)

UNIVERSITÉ CÔTE D'AZUR

Utilité des Licences

- ✓ Utilisées par les développeurs du noyau pour identifier
  - Problèmes venant de pilotes propriétaires, qu'ils n'essaieront pas de résoudre
- ✓ Permettent aux utilisateurs
  - Vérifier que leur système est à 100% libre
- ✓ Permettent aux distributeurs GNU/Linux
  - Vérifier la conformité à leur politique de licence

UNIVERSITÉ CÔTE D'AZUR

Ecriture d'un Module: Quelques Règles 1/2

- ✓ Includes C:
  - Impossible d'utiliser les fonctions de la bibliothèque C standard (`printf()`, `strcat()`, etc.).
  - La bibliothèque C est implémentée au dessus du noyau et non l'inverse
  - Linux a quelques fonctions C utiles comme `printk()`, qui possède une interface similaire à `printf()`
  - Donc, seul les fichiers d'entêtes du noyau sont autorisés.
- ✓ Virgule Flottante:
  - N'utilisez jamais de nombres à virgule flottante dans le code du noyau. Votre code peut être exécuter sur un processeur sans unité de calcul à virgule flottante (comme sur ARM)
  - L'émulation par le noyau est possible mais très lente

UNIVERSITÉ CÔTE D'AZUR

Ecriture d'un Module: Quelques Règles 2/2

- ✓ Portée:
  - Définissez tous vos symboles en local (`static`), hormis ceux qui sont exportés (`extern`): permet d'éviter la pollution de l'espace de nommage
- ✓ Consultez:
  - Dans le code source: `Documentation/CodingStyle`
- ✓ Il est toujours bon de connaître, voir d'appliquer, les règles de codage GNU:
  - http://www.gnu.org/prep/standards.html

UNIVERSITÉ CÔTE D'AZUR

Compilation d'un Module

- ✓ **Compilation en ligne de commande jusqu'en 2.4:**
  - `gcc -DMODULE -D__KERNEL__ -c hello.c`
- ✓ **Un Makefile à partir de 2.6**

```
# Makefile pour le module hello
obj-m := hello.o
KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)
default:
    $(MAKE) -C $(KDIR) M=$(PWD) modules
clean:
    $(MAKE) -C $(KDIR) M=$(PWD) clean
```

  - **Construit un fichier** `hello.ko`
  - **Si plusieurs fichiers à compiler pour un module**
    - `obj-m := file1.o file2.o file3.o`

UNIVERSITÉ CÔTE D'AZUR

Dépendances des Modules

- ✓ Les dépendances des modules n'ont pas à être spécifiées explicitement par le créateur du module.
- ✓ Elles sont déduites automatiquement lors de la compilation du noyau, grâce aux symboles exportés par le module:
  - ModuleB dépend de ModuleA si moduleB utilise un symbole exporté par moduleA.
- ✓ Les dépendances des modules sont stockées dans:
  - `/lib/modules/<version>/modules.dep.bin` (et `modules.dep` pour une version humainement lisible)
- ✓ Ce fichier est mis à jour (en tant que root) avec:
  - `depmod -a [<version>]`

UNIVERSITÉ CÔTE D'AZUR

Dépendances des Modules 2/2

- ✓ Pour être utilisable par un autre module, il faut exporter les symboles
  - Un symbole peut être une variable ou une fonction
- ✓ Utilisation de macros d'export dans le code
  - `EXPORT_SYMBOL(nom_du_symbole);`
  - `EXPORT_SYMBOL_NOVERS(nom_du_symbole);`
  - `EXPORT_SYMBOL_GPL(nom_du_symbole);`
- ✓ `EXPORT_SYMBOL*` déclare le symbole `extern`
- ✓ Pour ne rien exporter (comportement par défaut)
  - `EXPORT_NO_SYMBOL;`
- ✓ Les symboles sont placés dans le module symbole table du noyau

UNIVERSITÉ CÔTE D'AZUR

Utilisation des Symboles Statiques du Noyau

- ✓ En principe, seuls les symboles suivant sont exportés
  - Déclaré avec `EXPORT_SYMBOL` ou `extern`
  - Les symboles qui ont été prévus pour ça
    - Constitue l'API offerte par le noyau aux modules

- ✓ Que faire si on a besoin d'un symbole statique
  - Solution raisonnable 1 : s'en passer !
  - Solution raisonnable 2 : modifier le noyau
    - Ajouter `EXPORT_SYMBOL` ou `extern` et recompiler le noyau ...
  - Solution sale (mais efficace :-) : tricher
    - Rechercher l'adresse dans la table des symboles
      - `/proc/ksyms`
      - `/boot/System.mapXXX`

UNIVERSITÉ CÔTE D'AZUR

Exemple d'Utilisation de Symboles Statiques

- ✓ Hyp: on souhaite utiliser la variable **module_list**
  - Pb : symbole non exporté par le noyau
  - Solution à éviter :
    - Recherche symbole dans `System.map`

```
> grep module_list /boot/System.map-2.4.27
c01180d3 T get_module_list
c023edc0 D module_list
```

    - Déclaration du symbole dans le code du module

```
/* /boot/System.map nous donne l'adresse du symbole 'module_list'
* Comme module_list est un pointeur sur struct module, ce que nous
* récupérons est de type struct module **
*/
struct module **module_list_ptr = (struct module **)0xc023edc0;
```

  - Pourquoi « à éviter » au fait ?
    - Il faut vérifier (et modifier) le code du module à chaque recompilation du noyau (les symboles changent d'adresse)

UNIVERSITÉ CÔTE D'AZUR

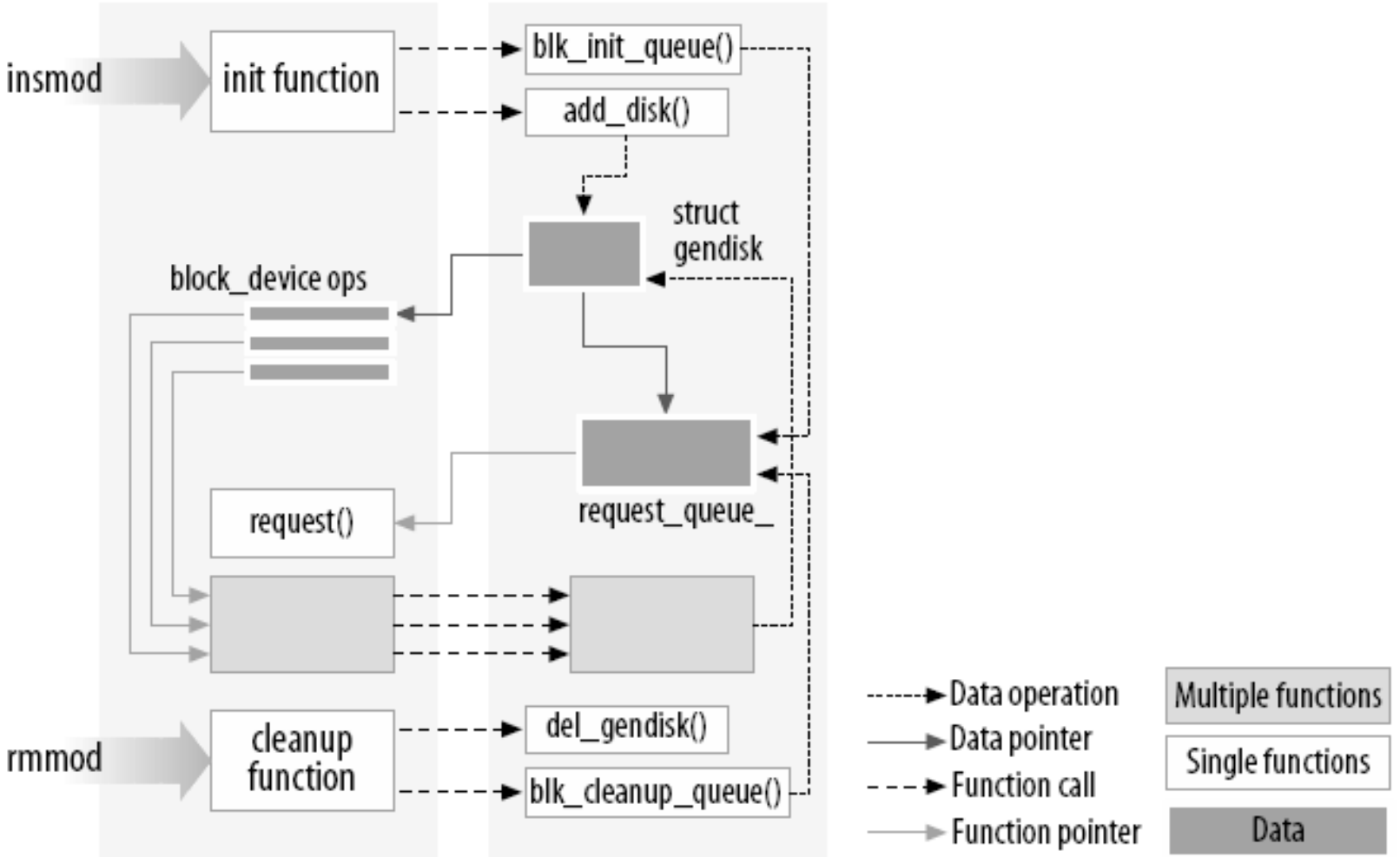
Commande pour la Gestion des Modules

- ✓ **Principales commandes:**
  - `insmod`: **charger un module en mémoire**
  - `modprobe`: **charge un module et toutes ses dépendances**
  - `rmmod`: **décharger un module de la mémoire**
  - `lsmod`: **voir les modules chargés (`cat /proc/modules`)**
  - `modinfo`: **voir les informations sur un module**
- ✓ **Exemple:**

```
~> lsmod
   Module                  Size  Used by    Not tainted
   mousedev                4308          1   (autoclean)
   input                   3648          0   (autoclean) [mousedev]
   usb-storage            70048          0   (unused)
   usb-uhci               23568          0   (unused)
   usbcore                63756          1   [usb-storage usb-uhci]
   eepro100               20276          1
   mii                     2464          0   [eepro100]
   vfat                   10892          0
   fat                    32792          0   [vfat]
   ext3                   82536          0   (autoclean)
   jbd                    42980          0   (autoclean) [ext3]
```

UNIVERSITÉ CÔTE D'AZUR

Modules Dynamiques – Schéma de Principe



UNIVERSITÉ CÔTE D'AZUR

Configuration d'un Module

- ✓ Certains modules ont besoin d'informations à l'exécution
  - Adresse hardware pour les I/O
  - Paramètres pour modifier le comportement (DMA...)
- ✓ 2 façons de les obtenir
  - Données par l'utilisateur
  - Auto détection
- ✓ Les paramètres sont passés au chargement à `insmod` ou `modprobe`
  - `insmod hello.ko param1=5 param2="Hello!"`
- ✓ Possibilité de les mettre dans un fichier de configuration (dépendant des distributions)
  - `Debian: /etc/modprobe.d/hello.conf`
    - `options hello param1=3 param2="Hello!"`

UNIVERSITÉ CÔTE D'AZUR

Passage de Paramètres à un Module ≥ 2.6

- ✓ Passage de paramètres à un module:
  - ✓ Le passages des paramètres sur la ligne de commande à un module ne s'effectue pas avec le mécanisme classique `argc/argv`.
  - – Utilisation de la macro: `module_param(name, type, permission)`
    - `name` = nom du paramètre
    - `type` = symbole indiquant son type
    - `permission` = exposition dans sysfs si différent de 0 (permission du fichier
  - – Types de variables supportés:
    - bool: boolean
    - charp: char * (chaîne de caractères)
    - short: short
    - ushort: unsigned short
    - int: integer
    - uint: unsigned integer
    - long: long
    - ulong: unsigned long
  - – Macro pour l'importation de tableaux:
    - `module_param_array(name, type, var pointer, permission)`

UNIVERSITÉ CÔTE D'AZUR

Module dépendant de la Version du Noyau

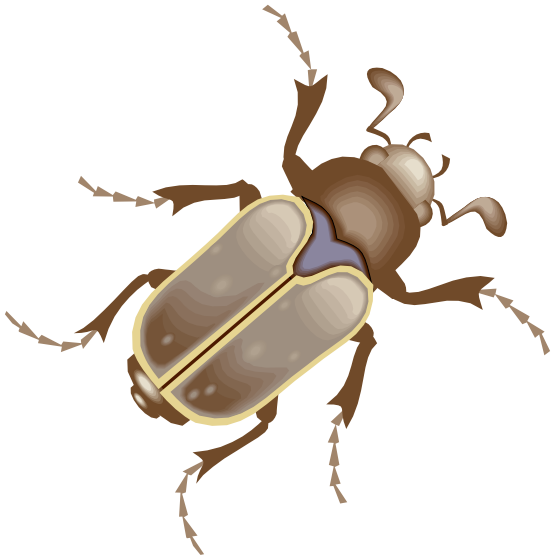
- ✓ Evolution du noyau
  - Généralement les appels systèmes (vu de l'espace utilisateur) restent les mêmes
    - Mais de nouveau appels systèmes peuvent apparaître
  - Les interfaces internes du noyau peuvent évoluer
    - Modification du numéro de version du noyau
- ✓ Donc on ne peut pas supposer que les interfaces restent les mêmes
  - Différences entre les version du noyau
  - Les modules peuvent dépendre de certaines versions d'interface
  - Des macros permettent de tester
    - Compare `LINUX_VERSION_CODE` to the macro `KERNEL_VERSION`
    - Défini pour Linux ≥ 2.0.35

UNIVERSITÉ CÔTE D'AZUR

Références Utiles

- ✓ The Linux Kernel Module Programming Guide
  - http://tldp.org/LDP/lkmpg/

UNIVERSITÉ
CÔTE D'AZUR



Débuguer un Dev Noyau

Où comment désinsectiser votre développement dans le noyau

UNIVERSITÉ CÔTE D'AZUR

Débugger en affichant

- ✓ Technique universelle de débuggage utilisée depuis les débuts de la programmation
  - Afficher des messages aux points clés du programme
    - Pour des programmes utilisateurs: `printf`
    - Mais dans l'espace noyau: `printk`
    - Dans les version plus récentes du noyau, utiliser les fonction pr_*()
      - `pr_emerg()`,`pr_alert()`, `pr_crit()`,`pr_err()`,`pr_warning()`, `pr_notice()`,`pr_cont()` et `pr_debug()`,...
- ✓ Affiché ou non dans le console ou `/var/log/messages`
  - Dépend de la priorité `linux/kernel.h` (voir slide suivant)
  - Plus le niveau est faible plus la priorité est élevée
  - Strings `"<[0-7]>"` concaténée au message de compilation
  - Si aucun niveau précisé, `DEFAULT_MESSAGE_LOGLEVEL` défini dans `kernel/printk.c`

UNIVERSITÉ CÔTE D'AZUR

Niveaux de log

- ✓ `# define KERN_EMERG "<0>"`
  - Messages urgents juste avant un système non utilisable
- ✓ `# define KERN_ALERT "<1>"`
  - Système nécessitant une action immédiate
- ✓ `# define KERN_CRIT "<2>"`
  - Situation critique (panne software ou hardware)
- ✓ `# define KERN_ERR "<3>"`
  - Situation d'erreur, problème hardware
- ✓ `# define KERN_WARNING "<4>"`
  - Sutiation problématique (pas un gros problème pour le système)
- ✓ `# define KERN_NOTICE "<5>"`
  - Situation normale mais significative
- ✓ `# define KERN_INFO "<6>"`
  - Message d'information (par exemple, nom assigné au hardware)
- ✓ `# define KERN_DEBUG "<7>"`
  - Message de débuggage

UNIVERSITÉ CÔTE D'AZUR

Oops

- ✓ Quand le noyau détecte un problème (opération illégale)
  - Message de panique du noyau: Oops
  - Tue le processus à l'origine du problème
  - Même si le système semble fonctionner correctement, effets de bords liés à l'arrêt de la tâche à l'origine du problème
- ✓ Un Oops contient
  - Une description textuelle
  - Le numéro de Oops
  - Le numéro de CPU
  - Les registres CPU
  - La pile d'appels
  - Les instructions que le CPU exécutait
- ✓ Ne contient aucun symbole, juste les adresses

UNIVERSITÉ
CÔTE D'AZUR

Exemple de Kernel Oops

UNIVERSITÉ CÔTE D'AZUR

ksymoops

- ✓ Aide pour le décryptage des messages oops
  - Converti les adresses et le code en informations utiles
- ✓ Facile à utiliser
  - Copier/coller le texte oops dans un fichier
- ✓ Nécessite les informations suivantes
  - System.map du noyau
  - Liste des modules (par défaut utilise `/proc/modules`)
  - Liste des symboles du kernel (`/proc/ksyms`)
  - Une copie de l'image du noyau
- ✓ Rend les adresses lisibles par des humains
  - `EIP; c88cd018 <[kbdb]__memp_fget+158/b54>`
- ✓ Exemple de ligne de commande
  - `ksymoops -no-ksyms -m System.map -v vmlinuz oops.txt`
- ✓ Voir `Documentation/oops-tracing.txt` et man `ksymoops`

UNIVERSITÉ CÔTE D'AZUR

Déboggeur Noyau

- ✓ Le noyau dispose de plusieurs déboggeurs
  - Outil d'aide au développement des pilotes ou fonctionnalités
  - Mode pas à pas comme un déboggeur classique
  - Longtemps refusé par Linus Towald (plus de 8 ans)
    - Pas l'objectif de faciliter la vie des développeurs noyau
- ✓ Le deux plus connus sont:
  - `kdb`
    - Debug au niveau code machine
    - Ne nécessite pas de travailler à distance (donc avec 2 machines)
    - http://oss.sgi.com/projects/kdb
    - Pas disponible pour les dernières versions du noyau (jusqu'à 2.6)
  - `kgdb`
    - Debug avec retour aux sources
    - Travail à distance (via une connexion série, Ethernet supprimé à l'intégration dans les sources du noyau pour simplifier)
    - http://kgdb.wiki.kernel.org
    - Disponible en tant que patch avant son intégration en 2.6.26
      - Option à activer dans Kernel Hacking