Linux embarqué
Retour d'expérience et temps réel

Denis Coupvent-Desgraviers

Introduction

- Bien connaître les bases d'un système d'exploitation
- Utilisation de GNU/Linux
- Bonnes connaissances en langage C
- Partir sur ces bases pour concevoir un système embarqué (maquette logicielle) sur x86
- Fort de cette expérience, application sur une architecture ARM
- Amélioration du projet avec l'aide d'un électronicien (temps réel ?)

PLAN : 1ère partie – Système d'exploitation

- Système d'exploitation
  - Historique des ordinateurs
  - Définition
  - Différents types
  - Typologie
  - Composition
  - Organisation générale
- Système d'exploitation Linux
  - Linux ?
  - Les processus
  - Les Threads
  - La Mémoire Partagée
  - Les Mutex
  - Les Sémaphores
  - L'ordonnanceur
  - Le multitâche
  - Le temps partagé
  - Le temps réel
- Conclusion
  - Améliorer la réactivité d'un système
  - Rendre temps réel Linux

PLAN : 2ème partie – retour d'expérience

- Création d'un système Linux
- Unité de Contrôle et de Communication
  - Création de la partition pour accueillir le système sur le serveur (<30 Mo)
  - Compilation et installation de Busybox
  - Copie des bibliothèques dynamiques utilisées par Busybox
  - Création des répertoires
  - Compilation et installation du noyau
  - Création des fichiers spéciaux pour les périphériques
  - Script de démarrage du système
  - Compilation + intégration du module GPIB dans le système de fichiers
  - Tests :
    - test de fonctionnement général
    - schéma de mesure
    - commande GPIB
    - Résultat du test
    - Résultat du test
- Adaptation sur ARM et perspectives du projet
- Vue générale sur cible ARM
- Conclusion

Plan : 3ème partie – Temps réel

- Temps réel en général
  - Définition
  - Tableau comparatif
  - Temps réel mou
  - Les limites du temps réel mou
- Outils de mesure des performances
- Exemples
- Conclusion
- Bibliographie

PLAN : 1ère partie – Système d'exploitation

- **Système d'exploitation**
  - **Historique des ordinateurs**
  - **Définition**
  - **Différents types**
  - **Typologie**
  - **Composition**
  - **Organisation générale**
- Système d'exploitation Linux
  - Linux ?
  - Les processus
  - Les Threads
  - La Mémoire Partagée
  - Les Mutex
  - Les Sémaphores
  - L'ordonnanceur
  - Le multitâche
  - Le temps partagé
  - Le temps réel
- Conclusion
  - Améliorer la réactivité d'un système
  - Rendre temps réel Linux

Historique des ordinateurs

- Les systèmes d'exploitations très liés à l'architecture des ordinateurs
- 1ère génération (1945-1955) : tubes à vide et tableaux d'interrupteurs
- 2ème génération (1955-1965) : transistors et systèmes par lots
- 3ème génération (1965-1980) : apparition de la multiprogrammation (utilisation CPU optimisée)
- 4ème génération (1980-aujourd'hui) : apparition des puces LSI (Large Scale Integration) –> ordinateur personnel

Définition 1/2

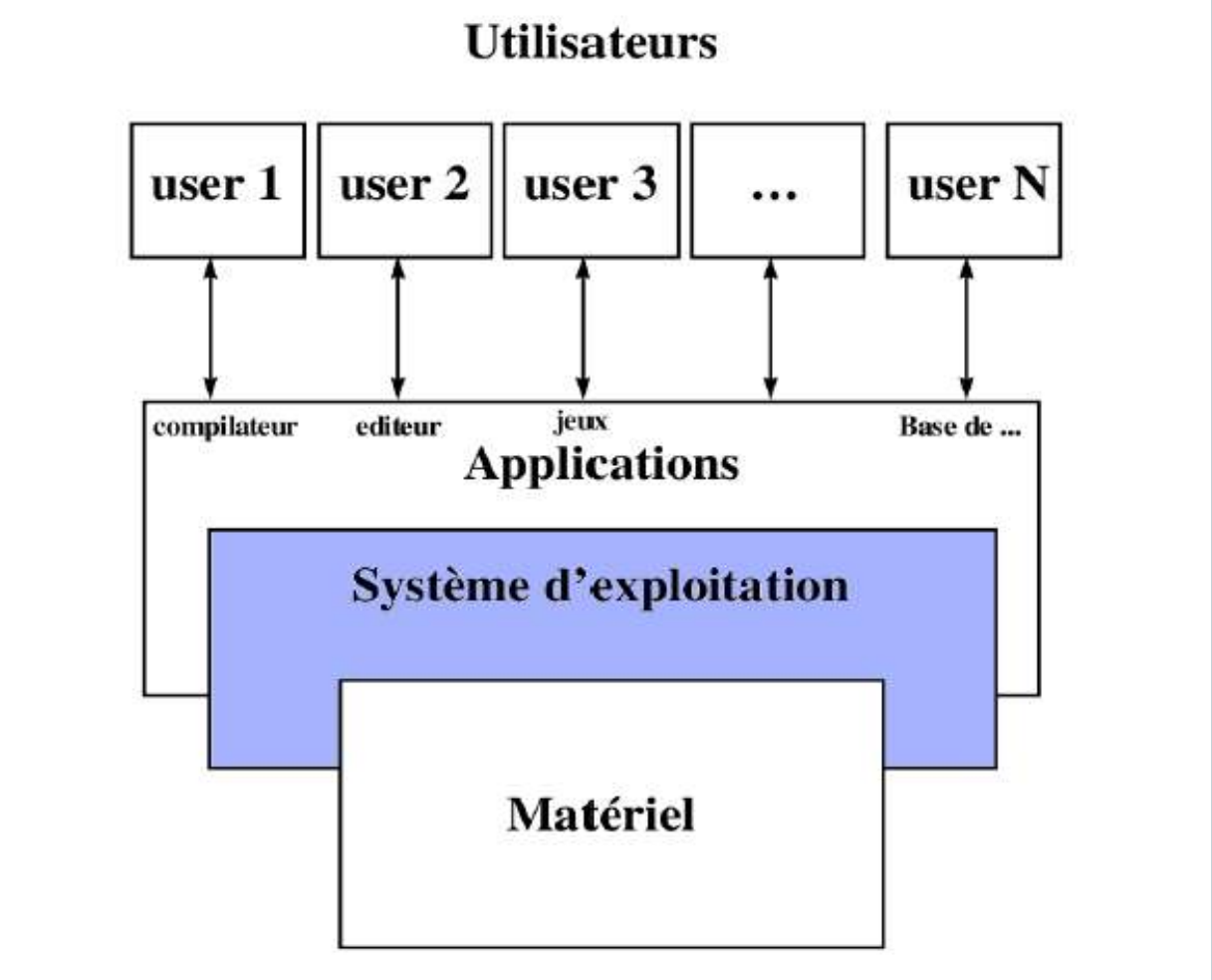
Le système d'exploitation (ensemble de programmes) s'intercale entre les logiciels et le matériel

- Rôle 1 : séparer les applications des spécificités et des limitations du matériel
- Rôle 2 : protéger le matériel des applications
- Rôle 3 : offrir une vue simple et cohérente au programmeur pour développer ses applications (API: Applications Programming Interface)

Pour réaliser les objectifs précédents, le système d'exploitation gère

- les processus (programmes)
- l'ordonnanceur
- la mémoire
- le système de fichiers
- les entrées/sorties
- le réseau
- l'interface graphique (accessoirement)

Définition 2/2



Différents types 1/3

- Il existe 9 types de systèmes d'exploitations:
- Système pour mainframe
- Système pour serveurs
- Système pour multiprocesseurs
- Système personnels
- Système temps réel
- Système embarqué

Différents types 2/3

- Système pour mainframe
  - Grosse machine (ex: IBM Mainframe AS 400)
  - Beaucoup de ressources avec 1 seul terminal (écran monochrome)
- Système pour serveur
  - Beaucoup d'utilisateurs
  - Contrainte de forte disponibilité
- Système multiprocesseurs
  - Plusieurs processeurs
  - Gestion concurrente des ressources

Différents types 3/3

- Système personnels
  - Interface conviviale
  - Ressources basiques
- Système temps réel
  - Respect des contraintes temporelles
  - Temps réel dur – temps réel mou
- Système embarqué
  - De plus en plus d'interface IHM conviviales
  - De plus en plus multiprogrammes != systèmes enfouis
  - Ressources matérielles limitées

Typologie 1/3

5 générations de système d'exploitation :

- Traitement par lot (1950)
- Multiprogrammés (1960)
- Temps partagé (1970)
- Temps réel (1975)
- Distribués (1990)

Typologie 2/3

- Traitement par lot (1950)
  - Cartes perforées (instructions + données) -> lecture séquentielle + synchrone des données (Batch)
  - Peu d'intervention du seul opérateur
  - Gros calculs, météo, statistiques, impôts…
- Multiprogrammés (multitâches) (1960)
  - Optimise l'utilisation du CPU
  - Première notion de scheduling (planification de l'utilisation du CPU cadencé par l'usage des périphériques)
  - Peu d'intervention du seul opérateur

Typologie 3/3

- Temps partagé (1970)
  - Plusieurs utilisateurs simultanés
  - Plusieurs programmes exécutés en temps égaux
  - Utilisation du swap
- Temps réel (1975)
  - Temps partagé (tous les OS) != temps réel (OS spécifiques ou adaptés)
  - L'information après acquisition et traitement reste encore pertinent
  - Ressources logicielles et/ou matérielles réservées
- Distribués (1990)
  - Les OS actuels sont préemptifs (!= coopératifs), multitâches, multiutilisateurs

Composition

- les processus (programmes) et leurs communications
- la mémoire
- l'ordonnanceur
- le système de fichiers
- Le contrôle d'accès
- Les utilitaires
- les entrées/sorties
- Le réseau
- L'interface graphique (accessoirement)

Organisation générale 1/3

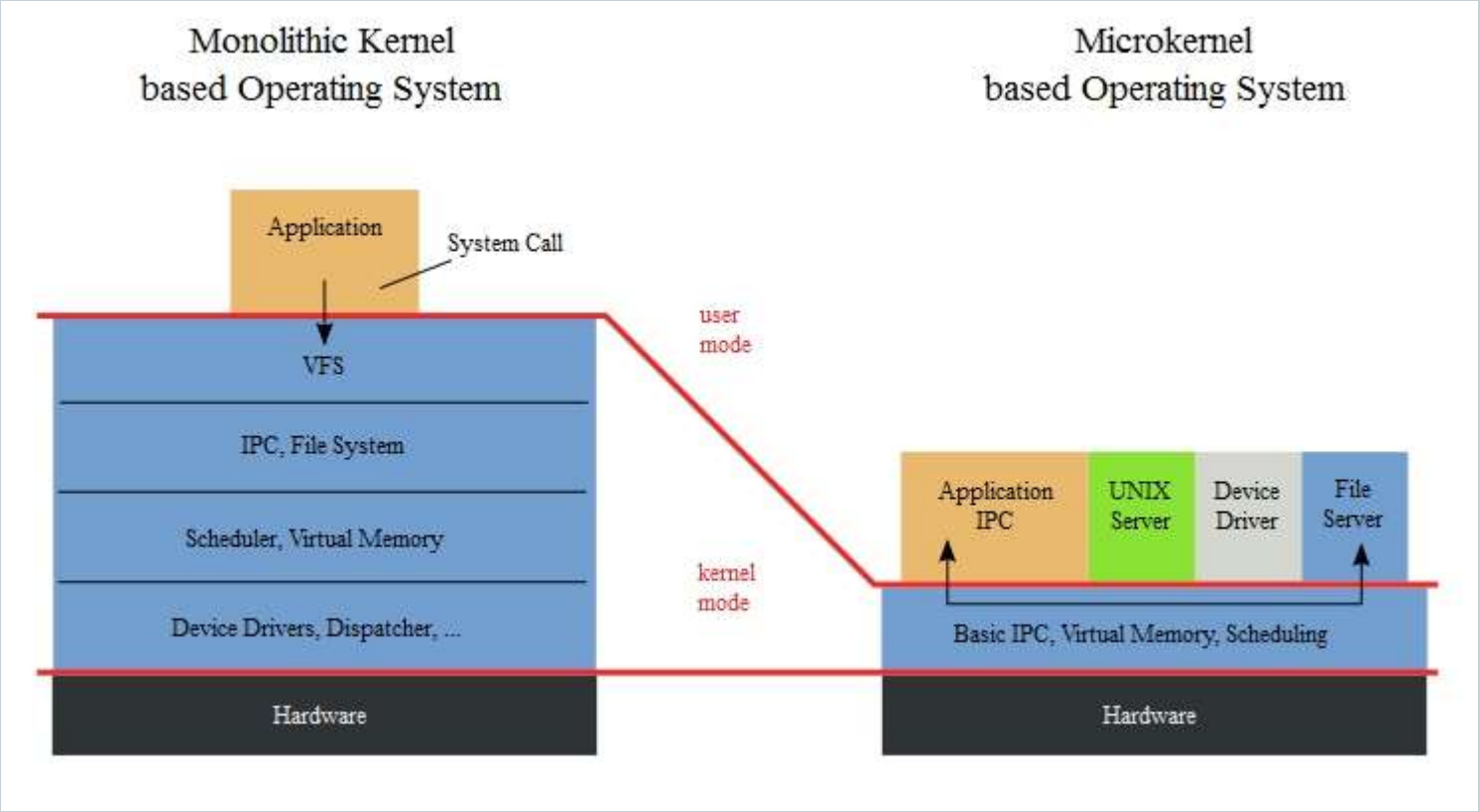
- Un système d'exploitation s'organise en différentes couches
- Couche haute : API, utilitaires, fenêtres
- Couche moyenne : mémoire, système de fichiers, pilote, réseau, etc…
- Couche basse : noyau

- Noyau : programme offrant une couche d'abstraction au système d'exploitation. Il s'intercale entre le matériel et l'ensemble des programmes qui forment la base minimale de l'OS. Le noyau occupe un espace mémoire protégé.

Organisation générale 2/3

- Le noyau oriente l'organisation générale du S.E.
  - Noyau monolithique: tout est dans le noyau (ex: famille Unix/Linux, FreeBSD)
  - Micro-noyau : seulement le strict minimum : ordonnanceur+mémoire virtuelle (ex: Minix, Mac OS X)
  - Noyau hybride (ex: Windows NT)
  - Exo-noyau rien n'est protégé (ex: Mac OS, AmigaOS)

Organisation générale 3/3



PLAN : 1ère partie – Système d'exploitation

- Système d'exploitation
  - Historique des ordinateurs
  - Définition
  - Différents types
  - Typologie
  - Composition
  - Organisation générale
- **Système d'exploitation Linux**
  - **Linux ?**
  - **Les processus**
  - **Les Threads**
  - **La Mémoire Partagée**
  - **Les Mutex**
  - **Les Sémaphores**
  - **L'ordonnanceur**
  - **Le multitâche**
  - **Le temps partagé**
  - **Le temps réel**
- Conclusion
  - Améliorer la réactivité d'un système
  - Rendre temps réel Linux

Linux ? 1/4

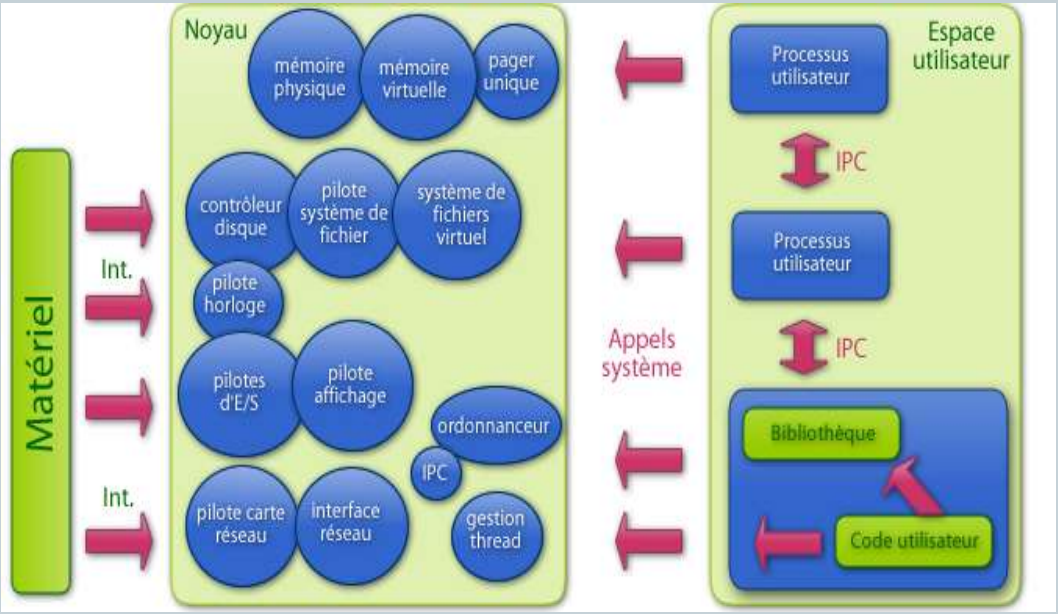
- Conçu par Linus Torvalds (étudiant finlandais)
- Linux <- dérivé de Minix <- dérivé de UNIX <- dérivé Multics
- Première version (0.01) annoncée sur le newsgroup comp.os.minix en septembre 1991
- Seulement 10 000 lignes de codes
- Aujourd'hui : version 3.11.6 -> 15 millions lignes de codes !
- Accélération des sorties des versions (développement mobile)

Linux ? 2/4

- Erreurs fréquentes :
  - Linux ne désigne que le **noyau**
  - Linux est souvent associé aux outils GNU (fondateur: Richard Stallman) d'où le nom de GNU/Linux (GNU : Gnu's Not Unix)
  - Systèmes avec les outils GNU mais un noyau différent :
  - GNU/Hurd, Solaris, etc...
- Systèmes Linux sans GNU : Android
- GNU/Linux : système d'exploitation :
  - multitâche
  - multi-utilisateur
  - temps partagé
  - Respecte la norme POSIX

Linux ? 3/4

- Structure du système GNU/Linux :



Linux ? 4/4

- L'API Linux offre des mécanismes de communication et de synchronisation:
  - entre threads (mutex, variables conditions…),
  - entre processus (files de messages, mémoires partagée, sémaphore, pipes…)
  - entre système distant (socket TCP/IP ou UDP/IP)

Les processus 1/2

- Un processus représente l'ensemble du programme en cours d'exécution et de ses données. Un utilisateur ne peut exécuter qu'un nombre limité de processus.
- Le système d'exploitation attribue un numéro unique (*pid*) à chaque processus.
- Le premier processus lancé par le noyau est le processus init. Il comporte le numéro 1. Tous les autres processus sont des « fork » (processus fils).
- La commande ***ps*** liste les processus en cours d'exécution
- La commande ***pstree*** donne une arborescence des processus.

Les processus 2/2

- 2 types de processus :

– Système: propriété du super-utilisateur (démons)

– Utilisateur: lancés par les utilisateurs

- Interruption d'un processus à la réception d'un signal

– A partir d'un programme (SIGINT, SIGQUIT)

– A partir d'un shell avec la commande:

kill num_signal pid_processus

- les processus sont protégés les uns des autres par le système. Ils peuvent communiquer entre eux par des signaux. La liste des signaux est donnée par la commande: ***kill -l***

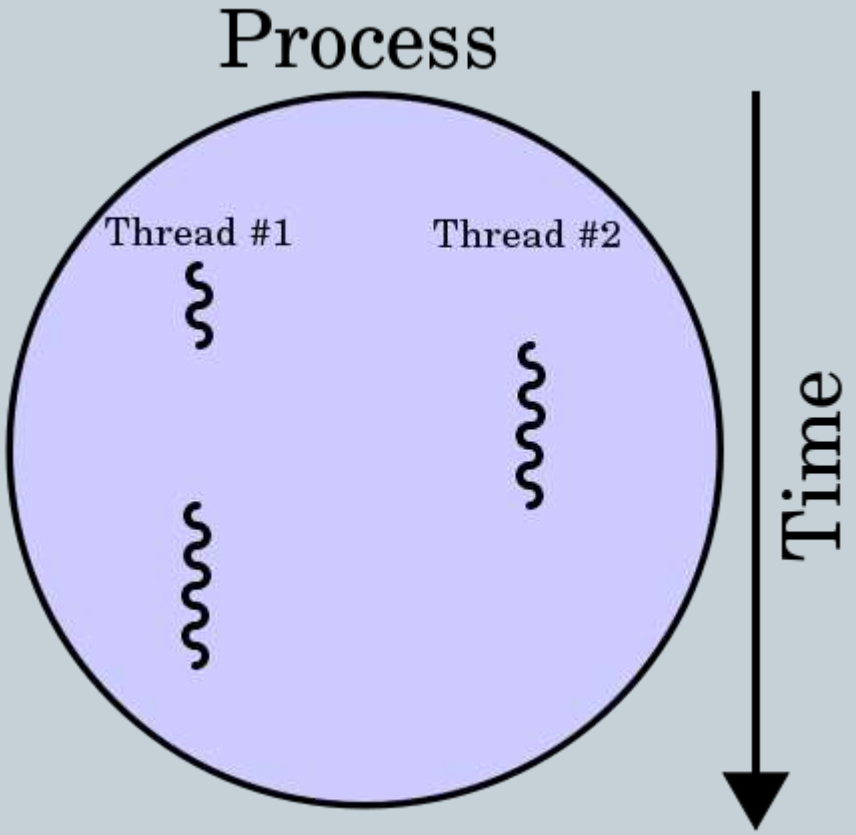
Les threads

•Un processus possède au moins 1 thread.

• Partage le même espace mémoire que le processus

•Chaque thread a sa propre pile d'appel

•Les threads permettent de paralléliser le déroulement du processus



La mémoire partagée

- Les processus s'exécutent en mémoire RAM.
- L'adressage logique (mémoire découpée en page) de la RAM est différente de l'adressage physique.
- Cette disposition permet de cloisonner l'espace mémoire alloué (pas de dépassement mémoire) et de protéger le système en cas de crash d'un processus.
- Cette gestion de la mémoire est remplie par la MMU (Memory Managing Unit).
- Conséquence importante : une portion physique de RAM peut apparaître dans l'espace virtuel de plusieurs processus à des adresses éventuellement différentes

Les mutex

- Mutex : MUTual EXclusion locks (verrous d'exclusion mutuelle)
- Un mutex est un verrouillage spécial qu'un seul thread peut utiliser pour réaliser une tâche dans une section critique
- Les mutex permettent de réguler la concurrence critique au niveau de la file de tâches entre thread en autorisant qu'un seul thread à accéder à cette file
- Les mutex évitent ainsi les problèmes d'accès concurrent à une ressource et aux données partagées

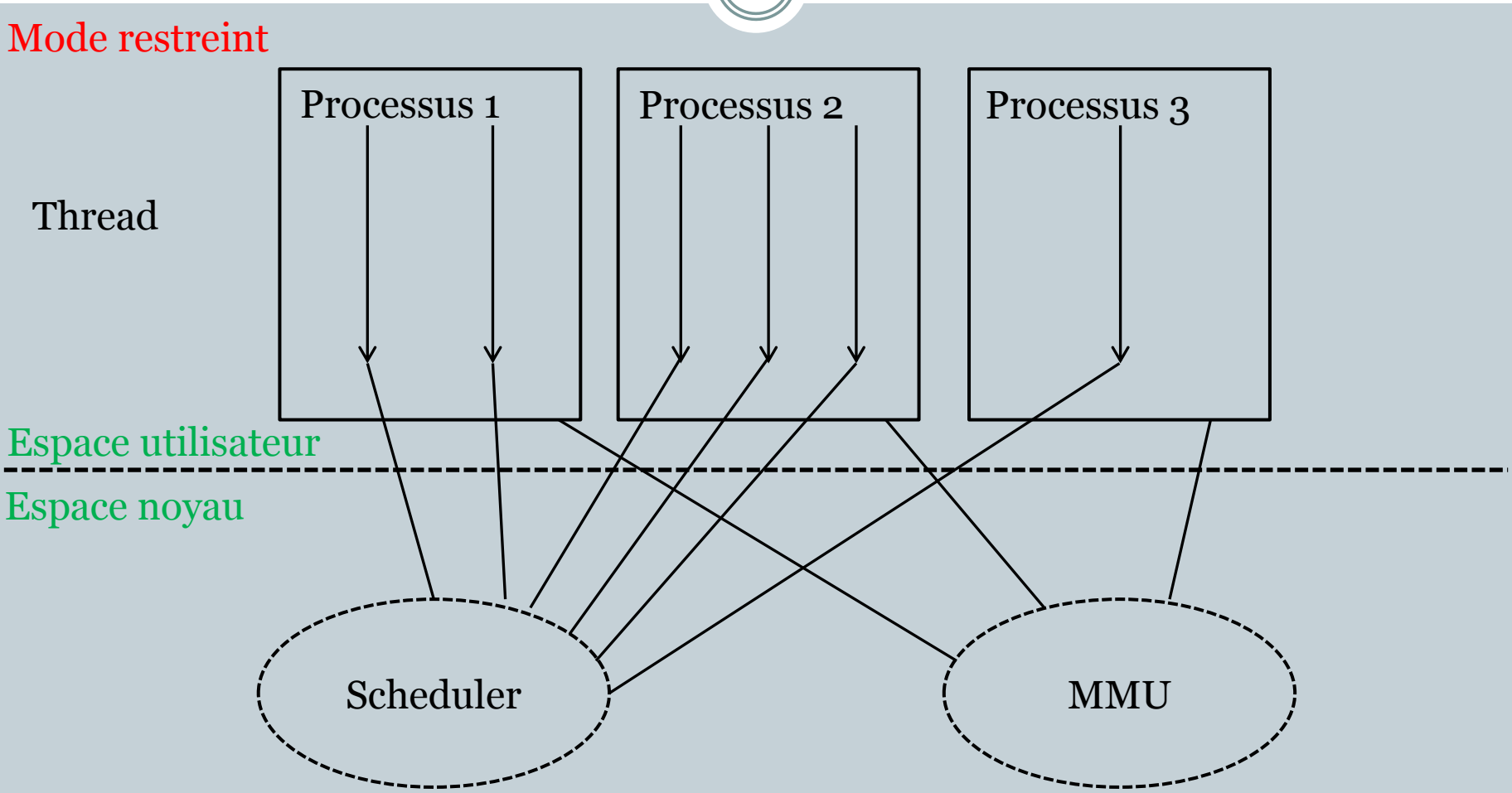
Les sémaphores

- Un sémaphore est un nombre entier positif ou nul représentant une quantité de ressources disponibles.
- Les sémaphores permettent une mise en attente lorsque cette ressource est indisponible.
- Les sémaphores sont des compteurs qui permettent la synchronisation de plusieurs threads
- Un sémaphore supporte deux opérations de base:
  - Une opération d'*attente* (wait), qui décrémente la valeur du sémaphore d'une unité. Si la valeur est déjà à zéro, l'opération est bloquante jusqu'à ce que la valeur du sémaphore redevienne positive
  - Une opération de *réveil* (post) qui incrémente la valeur du sémaphore d'une unité. Si le sémaphore était précédemment à zéro et que d'autres threads étaient en attente sur ce même sémaphore, un de ces threads est débloqué et son attente se termine (ce qui ramène la valeur du sémaphore à zéro).
- Il y a 1 différence entre les sémaphores POSIX et ceux implémentés sous Linux (sémaphore de processus) !

L’ordonnanceur

- L’actuel ordonnanceur CFS (Completely Fair Scheduler) type « best effort »
- Ordonnanceur basé sur une implémentation par arbres rouge-noir
- La priorité décroit en fonction de l'utilisation réelle du CPU
- Il permet de gérer de l'ordonnancement par groupes de processus (exemple: par utilisateur)

Le multitâche 1/4



Le multitâche 2/4

- Espace utilisateur : mode restreint = accès à certaines opérations d'entrées-sorties + certaines plages mémoires => si violation : levée d'une exception et reprise de contrôle par le noyau !

- Espace privilégié : mode noyau = tous les jeux d'instructions du CPU accessibles + toutes les opérations d'entrées-sorties + toutes les plages mémoires via la MMU (Memory Management Unit)

Le multitâche 3/4

- Espace utilisateur : toutes les applications s'y exécutent (mode utilisateur)
- Espace privilégié : exécution du code noyau (mode noyau)
- Passage du mode utilisateur au mode noyau dans 3 cas:
  - Appel système (sous-traitance d'1 tâche en mode noyau)
  - Exception détectée par le CPU
  - Interruption (périphérique)

Le multitâche 4/4

- Etat des tâches: commande ps maux

USER	PID	%CPU	%MEM	STAT	START	TIME	COMMAND
root	1	0.0	0.0	-	Oct02	0:06	/sbin/init
root	-	0.0	-	Ss	Oct02	0:06	-
root	2	0.0	0.0	-	Oct02	0:00	[kthreadd]
root	-	0.0	-	S	Oct02	0:00	-
[…]							
dc	6446	0.0	0.0	-	15:04	0:00	gnome-pty-helper
dc	-	0.0	-	S	15:04	0:00	-
dc	6447	2.1	0.1	-	15:04	0:00	bash
dc	-	2.1	-	Ss	15:04	0:00	-
[…]							

Le temps partagé 1/5

- Définition : mode d'ordonnancement permettant d'allouer un temps équitable à toutes les tâches donnant ainsi l'illusion d'un fonctionnement simultané sur un même processeur. Temps partagé ≠ Multitâche préemptif
- 3 cas de figures où le CPU interrompt l'exécution d'1 tâche pour en activer une autre:
  - Volontairement en invoquant 1 appel système bloquant
  - Involontairement avec l'avènement d'une interruption dont la tâche réveillée est plus prioritaire que celle en cours d'exécution
  - Temps CPU alloué à une tâche est atteint

Le temps partagé 2/5

- Ordonnancement temps partagé: même temps d'exécution pour tous les processus. Possibilité d'affiner la courtoisie (≠ priorité) des processus par la commande « nice »de -20 (priorité haute) à 19 (priorité basse). Par défaut : 0
- Exemples:
  - nice -n -20 ./nom_de_executable (processus très prioritaire)
  - nice -n 5 ./nom_de_executable
- En programmation C, ce sont les fonctions :
  - Int getpriority (int type, int ident);
  - Int setpriority(int type, int ident, int nice);
- Pour passer d'une tâche à une autre : changement de contexte

Le temps partagé 3/5

- Limitations de l'ordonnancement temps partagé:
  - Le noyau linux actuel nous permet de mesurer l'heure avec une précision de l'ordre de la microseconde
  - Nous obtenons des timers logiciels dont la période se mesure en dizaine ou centaine de microsecondes
  - Dans le cas d'un comportement périodique, si le système est chargé => mesure moins efficace
  - L'ordonnancement temps partagé induit une fluctuation (jitter) des timers et des préemptions inévitables des tâches.

Le temps partagé 4/5

- MESURE D'UNE OPERATION CRITIQUE : mesure 30 fois l'heure en boucle et affichage des résultats APRES la mesure
- Utilisation de la fonction: `gettimeofday()`

- Précision de mesure du temps avec la fonction time() avec **mon vieux portable x86 (Dual core, 1.86 Ghz, frequence bus 533Mhz)** :

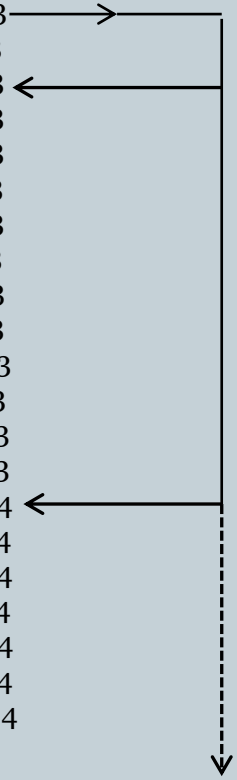
- 0 : 1384097758.182905
- 1 : 1384097758.182908
- 2 : 1384097758.182911
- 3 : 1384097758.182913
- 4 : 1384097758.182916
- 5 : 1384097758.182918
- 6 : 1384097758.182921
- 7 : 1384097758.182923
- 8 : 1384097758.182926
- 9 : 1384097758.182928
- 10 : 1384097758.182931
- 11 : 1384097758.182933
- 12 : 1384097758.182936
- 13 : 1384097758.182938
- 14 : 1384097758.182941
- 15 : 1384097758.182944
- 16 : 1384097758.182946
- 17 : 1384097758.182949
- 18 : 1384097758.182951
- 19 : 1384097758.182954
- 20 : 1384097758.182956

Saut de 2 à 3microsecondes

Le temps partagé 5/5

- MESURE D'UNE OPERATION CRITIQUE :
- Précision de mesure du temps avec la fonction time() avec **une carte ARM (Mono-coeur, Frq :180 Mhz)** :

- 0 : 1384268783.708923
- 1 : 1384268783.708923
- 2 : 1384268783.708953
- 3 : 1384268783.708953
- 4 : 1384268783.708953
- 5 : 1384268783.708953
- 6 : 1384268783.708953
- 7 : 1384268783.708953
- 8 : 1384268783.708953
- 9 : 1384268783.708953
- 10 : 1384268783.708953
- 11 : 1384268783.708953
- 12 : 1384268783.708953
- 13 : 1384268783.708953
- 14 : 1384268783.708984
- 15 : 1384268783.708984
- 16 : 1384268783.708984
- 17 : 1384268783.708984
- 18 : 1384268783.708984
- 19 : 1384268783.708984
- 20 : 1384268783.708984



Saut de 30 microsecondes !

PLAN : 1ère partie – Système d'exploitation

- Système d'exploitation
  - Historique des ordinateurs
  - Définition
  - Différents types
  - Typologie
  - Composition
  - Organisation générale
- Système d'exploitation Linux
  - Linux ?
  - Les processus
  - Les Threads
  - La Mémoire Partagée
  - Les Mutex
  - Les Sémaphores
  - L'ordonnanceur
  - Le multitâche
  - Le temps partagé
  - Le temps réel
- **Conclusion**
  - **Améliorer la réactivité d'un système**
  - **Rendre temps réel Linux**

Conclusion : améliorer la réactivité d'un système

- On peut améliorer la réactivité d'un système à différentes stimulations :
  - Temps de réponse aux **interruptions matérielles** : dialogue avec des dispositifs externes, liens de communication, pilotage de processus industriels, etc.
  - Granularité et précision des **timers logiciels** : déclenchement d'événements cadencés régulièrement, émission de données dans des créneaux temporels précis, etc.
  - **Priorités des tâches** : précédence de réponse à un événement externe, réactivité de l'interface utilisateur, traitements en tâche de fond…

Conclusion: rendre temps réel Linux

- En dégraissant le noyau original (vanilla) d'options inutiles.
- En activant/désactivant quelques options pertinentes du noyau:
  - Désactiver « *Tickless System* »
  - Activer le support « *High Resolution Timer* »
  - Activer « *Preemptible Kernel (Low-Latency Desktop)* »

- En patchant le noyau initial par Linux-rt
  - avantages : même API et même noyau => bonne portabilité, apparition de l'option « Preemptible kernel » dans le menu de compilation
  - Inconvénient : temps réel mou

- En dédiant un noyau (nanokernels) aux tâches temps réel :
  - RTLinux (très ancien)
  - RTAI (ancien)
  - Xenomai (à utiliser -> très bien maintenu ; API dédié au temps réel => apprentissage plus long)

PLAN : 2ème partie – retour d'expérience

- Création d'un système Linux
- Unité de Contrôle et de Communication
  - Création de la partition pour accueillir le système sur le serveur (<30 Mo)
  - Compilation et installation de Busybox
  - Copie des bibliothèques dynamiques utilisées par Busybox
  - Création des répertoires
  - Compilation et installation du noyau
  - Création des fichiers spéciaux pour les périphériques
  - Script de démarrage du système
  - Compilation + intégration du module GPIB dans le système de fichiers
  - Tests :
    - test de fonctionnement général
    - schéma de mesure
    - commande GPIB
    - Résultat du test
    - Résultat du test
- Adaptation sur ARM et perspectives du projet
- Vue générale sur cible ARM
- Conclusion

PLAN : 2ème partie – retour d'expérience

- **Création d'un système Linux**
- Unité de Contrôle et de Communication
  - Création de la partition pour accueillir le système sur le serveur (<30 Mo)
  - Compilation et installation de Busybox
  - Copie des bibliothèques dynamiques utilisées par Busybox
  - Création des répertoires
  - Compilation et installation du noyau
  - Création des fichiers spéciaux pour les périphériques
  - Script de démarrage du système
  - Compilation + intégration du module GPIB dans le système de fichiers
  - Tests :
    - test de fonctionnement général
    - schéma de mesure
    - commande GPIB
    - Résultat du test
    - Résultat du test
- Adaptation sur ARM et perspectives du projet
- Vue générale sur cible ARM
- Conclusion

Retour d'expérience

- Problème de pérennité (pilote de périphérique lié à une machine - OS) et de coût exorbitant de National Instrument
- Manque de compétence en système embarqué au laboratoire (2009)
- Etude faite sur le tas avec des connaissances initiales d'ASR
- Présentation orientée pédagogie avec la création pas à pas d'un système embarqué Linux sur x86

Création d'un système Linux

- Tutoriel sur carte x86
  - du moins vers le plus !!

1. Création de la partition pour accueillir le système sur le serveur(<30 Mo) + utilisation de Grub
2. Compilation et installation de Busybox
3. Copie des bibliothèques dynamiques utilisées par Busybox
4. Création des répertoires
5. Compilation et installation du noyau
6. Création des fichiers spéciaux pour les périphériques
7. Script de démarrage du système
8. Compilation + intégration du module GPIB dans le système de fichiers
9. Test du système

PLAN : 2ème partie – retour d'expérience

- Création d'un système Linux
- **Unité de Contrôle et de Communication**
  - **Création de la partition pour accueillir le système sur le serveur (<30 Mo)**
  - **Compilation et installation de Busybox**
  - **Copie des bibliothèques dynamiques utilisées par Busybox**
  - **Création des répertoires**
  - **Compilation et installation du noyau**
  - **Création des fichiers spéciaux pour les périphériques**
  - **Script de démarrage du système**
  - **Compilation + intégration du module GPIB dans le système de fichiers**
  - **Tests :**
    - **test de fonctionnement général**
    - **schéma de mesure**
    - **commande GPIB**
    - **Résultat du test**
    - **Résultat du test**
- Adaptation sur ARM et perspectives du projet
- Vue générale sur cible ARM
- Conclusion

Unité de Contrôle et de Communication

1/9. Création de la partition pour accueillir le système sur le serveur (<30 Mo)

- Création de la partition :

```
fdisk /dev/hda
```

- Formatage de la partition :

```
mk2efs -j /dev/hda4
```

- Montage de la partition :

```
mkdir /mnt/emb
mount /dev/hda4 /mnt/emb
```

- Montage systématique de la partition au démarrage :

=> ajout de la ligne suivante au fichier **`/etc/fstab`** de notre distribution:

```
/dev/hda4          /mnt/emb ext3     defaults 0          0
```

Unité de Contrôle et de Communication

2/9 Compilation et installation de Busybox

- Compilation :

```
make menuconfig
make
```

- Installation :

```
make install
```

Résultat :

```
root@pmcpcdc-debian:/home/dc/Desktop/busybox-1.00 # ls -la /rootfs/
total 20
drwxr-xr-x  5 root root 4096 sep 14 16:00 .
drwxr-xr-x 28 root root 4096 sep 13 23:14 ..
drwxr-xr-x  2 root root 4096 sep 14 16:00 bin
lrwxrwxrwx  1 root root   11 sep 14 16:00 linuxrc -> bin/busybox
drwxr-xr-x  2 root root 4096 sep 14 16:00 sbin
drwxr-xr-x  4 root root 4096 sep 14 16:00 usr
```

Unité de Contrôle et de Communication

3/9 Copie des bibliothèques dynamiques utilisées par Busybox

- Outil mklibs :

```
cd /mnt/emb
mkdir lib
mklibs -v -d lib bin/*
```

Unité de Contrôle et de Communication

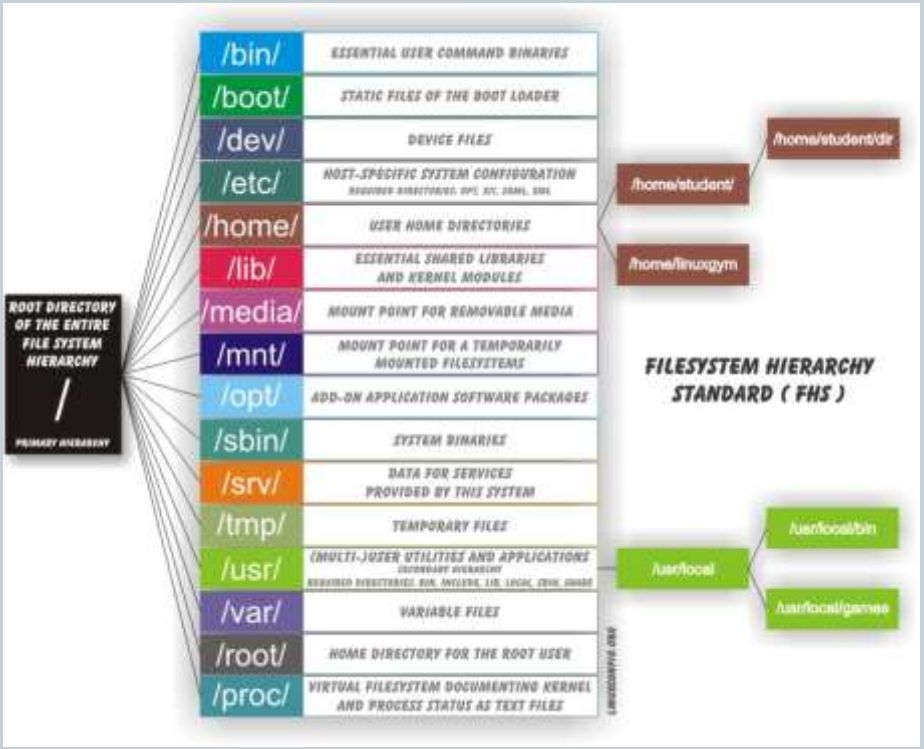
4/9 Création des répertoires

```
cd /mnt/emb

mkdir -p boot dev etc home mnt/floppy mnt/gpib proc root
sys share tmp usr var/log var/run

chmod 1777 tmp
```

/mnt/emb ⟷



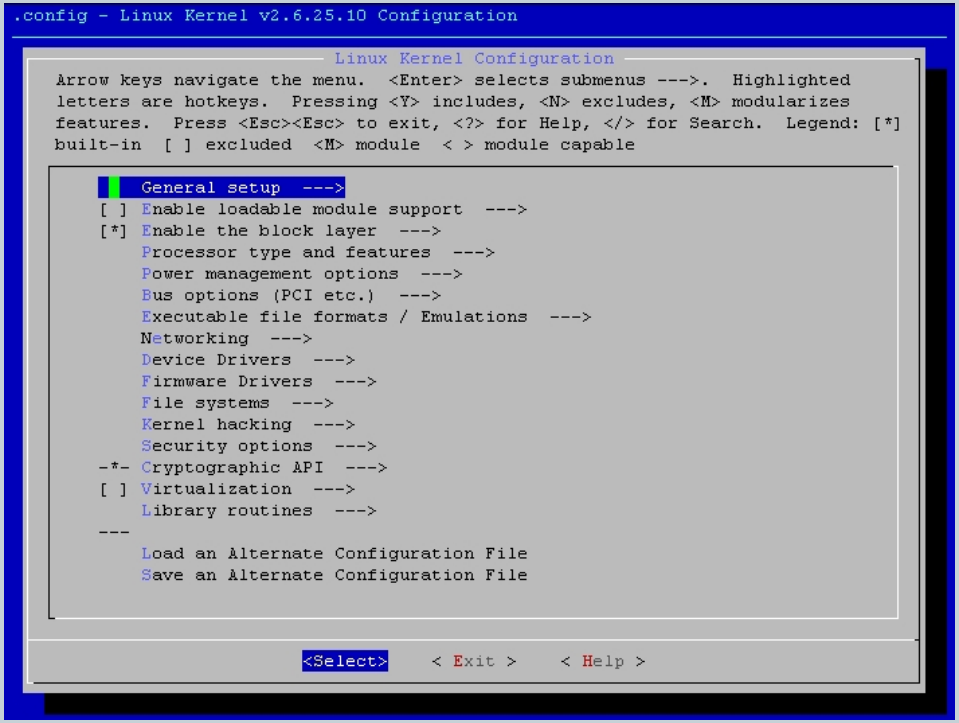
Unité de Contrôle et de Communication

5/9 Compilation et installation du noyau

- Compilation :

```
export INSTALL_PATH=/mnt/emb/boot
export INSTALL_MOD_PATH=/mnt/emb

make menuconfig
make
```



- Installation :

```
make install
```

Unité de Contrôle et de Communication

6/9 Création des fichiers spéciaux pour les périphériques

- Utilisation de la commande MAKEDEV :

Copie de l'exécutable dans la partition dédiée à l'embarqué :

```
cp /sbin/MAKEDEV /mnt/emb/dev/
```

Déplacement vers le répertoire où l'exécutable a été copié :

```
cd /mnt/emb/dev/
```

Génération des fichiers spéciaux :

```
./MAKEDEV generic
```

Unité de Contrôle et de Communication

7/9 Script de démarrage du système

- Après le lancement du processus **init** par le noyau celui-ci via le fichier `/mnt/emb/etc/inittab` lance l'exécutable `/mnt/emb/etc/init.d/rcS`

Ce fichier permet d'initialiser le système et de démarrer un certain nombre de processus

- Extrait du script rcS :

[…]

```
echo "Mounting proc et sys filesystem: " ;
mount -t proc /proc /proc
mount -t sysfs none /sys
# Remount the root filesystem in read-write mode
echo "Remounting root device with read-write enabled"
/bin/mount -w -n -o remount /
```

[…]

Unité de Contrôle et de Communication

8/9 Compilation + intégration du module GPIB dans le système de fichiers

- Projet Libre et toujours actif à l'adresse :
http://linux-gpib.sourceforge.net

- Compilation du module par :

```
./configure & make & make install
```

- Installation du module et de ses librairies :

```
cp ni_usb_gpib.ko /mnt/emb/lib/modules/2.6.25.10/gpib

cp -pr /lib/libpthread* /mnt/emb/lib/
```

- Unique fichier de configuration **`/etc/gpib.conf`** très détaillé

- Lecture de la configuration de notre interface en lançant l'exécutable **`gpib_conf`**

- Écriture des programmes de lecture et d'écriture pour notre interface GPIB

Unité de Contrôle et de Communication

9/9 : Tests 1/5 : test de fonctionnement général

Test du système

```
loading GPIB modules...

<<<  gpib_common module loading success  >>>

<<<  ni_usb_gpib  module loading success  >>>

<<<  record configuration from file gpib.conf  >>>

<<< parametre from file gpib.conf recorded with success  >>>


Distrib glibc + dropbear 2.6.25.10 ttyS0


pmctry.memoire.fr login:
  Aide : CTRL-A Z  | 19200 8N1 | NOR | Minicom 2.1    | VT102 | D+Rconnect+R
```

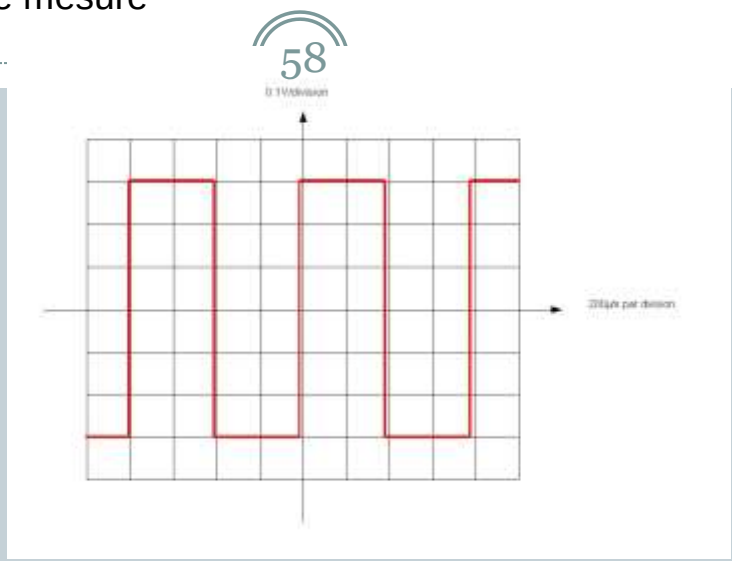
Test du module GPIB

```
root@pmctry:~# lsmod
Module                  Size  Used by    Not tainted
ni_usb_gpib            28928  0
gpib_common            32388  1 ni_usb_gpib
ipv6                  240036  10
```

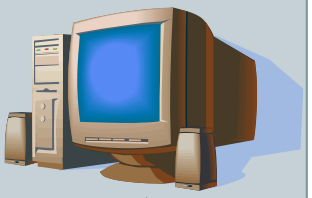
Unité de Contrôle et de Communication

9/9 : Test 2/5 : schéma de mesure

Signal carré périodique de fréquence : 1250Hz
Amplitude : 0.6V



PC hôte



Flot de données utilisateur par Ethernet et/ou port série

Système embarqué X86

Convertisseur USB/GPIB

Sonde

Oscilloscope

crochet

GPIB

USB

Unité de Contrôle et de Communication

9/9 : Test 3/5 : commande GPIB

Envoi des commandes GPIB :

Programmes écrits en C

Commandes envoyées à l'oscilloscope

```
gpib_config
gpib_write 1 "*cls"
gpib_write 1 "*rst"
gpib_write 1 "autoscale"
gpib_write 1 ":waveform:source channel1"
gpib_write 1 ":acquire:type normal"
gpib_write 1 ":acquire:complete 100"
gpib_write 1 ":waveform:points 1000"
gpib_write 1 ":waveform:format byte"
gpib_write 1 ":digitize channel1 "
```

Adresse primaire de l'oscilloscope

Réception des données :

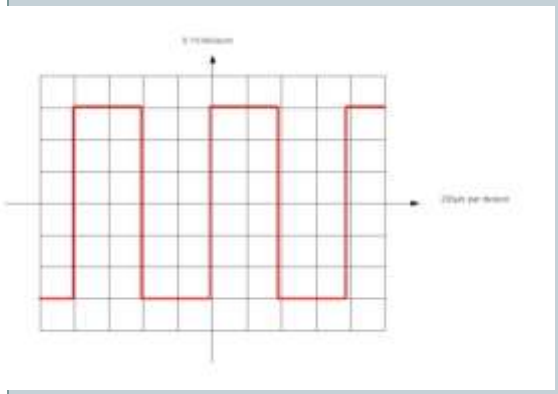
```
gpib_ read_PREAMBLE   1 100
gpib_read_bin_DATA    1 1000
```

Unité de Contrôle et de Communication

9/9 : Test 4/5 Résultat du test

Extrait des Data brutes récupérées de l'oscilloscope :

Signal initial :

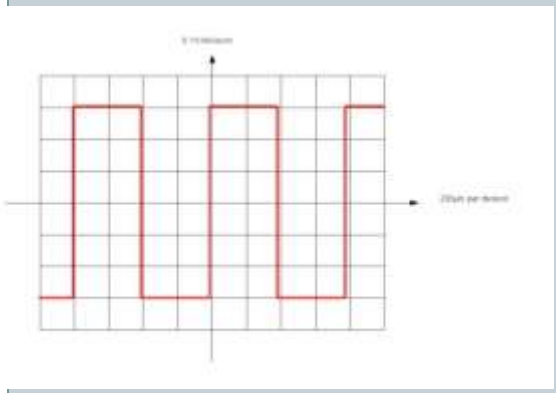


2d 2d 2e 2d 2e 2d 2f 2e 2e 2e 2e 2e 2e 2e 2e 2e 2e 2f 2e 2f 2f 2f 2e 2e 2f 2e
2e 2e 2f 2f 2f 2f 2e 2e 2f 2e 2f 2f 2e 2f 2f 2e 2f 2f 2f 2f 2f 2f 2f 2f 2f 2f 2f
2e 2f 2e 2f 2f 30 2f 30 2f 2f 2f 2f 2f 2f 2f 2f 30 2f 2f 2f 2f 2f 30 2f 2f 2f 2f 2f
2f 2f 2f 30 30 2f 2f 30 2f 30 30 2f 30 30 98 df df de df df de de de de de de
dd dc dd dd dc dc dc dc db db db db db da da da d9 d9 d9 d9 d9 d8 d9 d8
d9 d8 d8 d8 d8 d8 d7 d7 d7 d7 d7 d6 d7 d7 d6 d6 d7 d6 d6 d5 d5 d6 d5
d5 d6 d5 d5 d4 d5 d5 d5 d5 d5 d4 d5 d4 d5 d4 d4 d4 d3 d4 d2 d3 d3 d4
d3 d4 d4 d3 d3 d3 d2 d3 d3 d3 d3 d3 d3 d3 d3 d3 d3 d2 d3 d3 d3 d2 d2
d2 d2 d2 d2 d2 d2 d2 d2 d2 d2 d2 d2 d2 d2 d2 d2 d2 d2 d2 d2 d2 d2 d1 d0
d1 d0 d0 d2 d2 d1 d1 d1 d2 d1 d1 d1 d2 d1 d1 d1 d2 d1 d1 d2 d1 d1 d1
d1 d1 d0 d2 d1 d0 d2 d1 d1 d2 d1 d0 d1 d1 d1 d1 d2 d2 d2 d1 d1 d1 d1
d0 d1 d2 d0 d1 d1 d0 d0 d0 d1 d1 d1 d1 d1 d0 d1 d1 d1 d1 d1 d1 d0 d2
d1 d1 d1 d1 d1 d1 3a 21 21 21 21 21 21 22 22 22 23 22 23 23 23 24 24
24 25 25 25 24 26 25 25 25 26 26 26 27 27 26 27 28 28 28 28 28 28 28
29 28 2a 29 29 29 2a 29 2b 29 29 2a 2a 2a 2b 2b 2a 2b 2b 2b 2b 2b…

Unité de Contrôle et de Communication

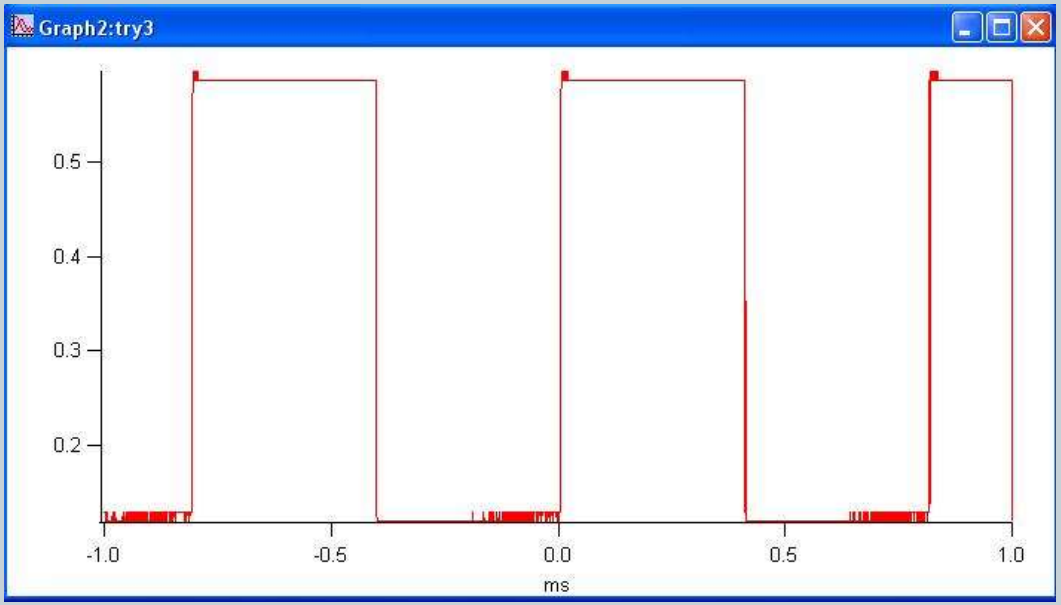
9/9 : Test 5/5 Résultat du test

Signal initial :



Après post-traitement : Résultat sous Igor

Volts



0.6 V

Temps

Période: 0.8ms ->1250Hz

PLAN : 2ème partie – retour d'expérience

- Création d'un système Linux
- Unité de Contrôle et de Communication
  - Création de la partition pour accueillir le système sur le serveur (<30 Mo)
  - Compilation et installation de Busybox
  - Copie des bibliothèques dynamiques utilisées par Busybox
  - Création des répertoires
  - Compilation et installation du noyau
  - Création des fichiers spéciaux pour les périphériques
  - Script de démarrage du système
  - Compilation + intégration du module GPIB dans le système de fichiers
  - Tests :
    - test de fonctionnement général
    - schéma de mesure
    - commande GPIB
    - Résultat du test
    - Résultat du test
- **Adaptation sur ARM et perspectives du projet**
- Vue générale sur cible ARM
- Conclusion

Unité de Contrôle et de Communication

Adaptation sur ARM et perspectives du projet

- Mise en œuvre du protocole de conception du système sur carte KB9200 de Kwikbyte (ARM 9) :

• Carte supportée nativement par les noyaux Linux de la famille 2.6
• Installation d'une chaîne de compilation croisée
• Compilation avec les options pour make : ARCH=arm CROSS_COMPILE=arm-linux-
pour configure : --build=i686-pc-linux-gnu –host=arm-linux

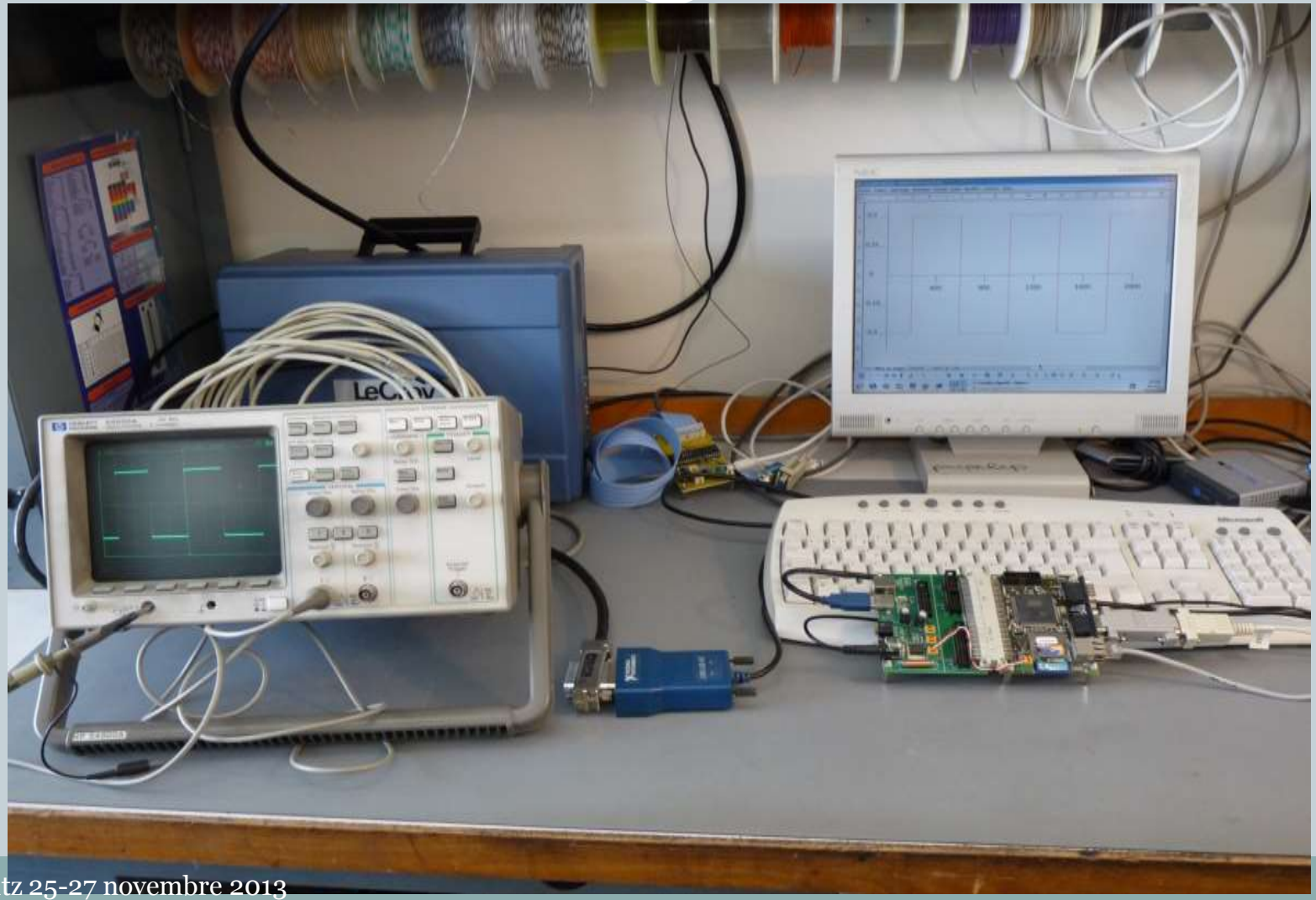
- Mise en place du Temps réel (en cours)
- Utilisation de la bibliothèque µClibc pour alléger l'empreinte mémoire
- Mesure en continu -> Pilotage par **CORBA** (**C**ommon **O**bject **R**equest **B**roker) ?

PLAN : 2ème partie – retour d'expérience

- Création d'un système Linux
- Unité de Contrôle et de Communication
  - Création de la partition pour accueillir le système sur le serveur (<30 Mo)
  - Compilation et installation de Busybox
  - Copie des bibliothèques dynamiques utilisées par Busybox
  - Création des répertoires
  - Compilation et installation du noyau
  - Création des fichiers spéciaux pour les périphériques
  - Script de démarrage du système
  - Compilation + intégration du module GPIB dans le système de fichiers
  - Tests :
    - test de fonctionnement général
    - schéma de mesure
    - commande GPIB
    - Résultat du test
    - Résultat du test
- Adaptation sur ARM et perspectives du projet
- **Vue générale sur cible ARM**
- Conclusion

Unité de Contrôle et de Communication

Vue générale sur cible ARM



PLAN : 2ème partie – retour d'expérience

- Création d'un système Linux
- Unité de Contrôle et de Communication
  - Création de la partition pour accueillir le système sur le serveur (<30 Mo)
  - Compilation et installation de Busybox
  - Copie des bibliothèques dynamiques utilisées par Busybox
  - Création des répertoires
  - Compilation et installation du noyau
  - Création des fichiers spéciaux pour les périphériques
  - Script de démarrage du système
  - Compilation + intégration du module GPIB dans le système de fichiers
  - Tests :
    - test de fonctionnement général
    - schéma de mesure
    - commande GPIB
    - Résultat du test
    - Résultat du test
- Adaptation sur ARM et perspectives du projet
- Vue générale sur cible ARM
- **Conclusion**

Conclusion

- Le cheminement de la solution du développement sur architecture x86 et ARM est proche
- Le résultat attendu est bon. Adaptation de la solution en temps réel en cours
- Pour éviter de « refaire » un système embarqué Linux « from scratch », étude de l'adaptation du projet à partir des projets Buildroot et OpenEmbedded.

Plan : 3ème partie – Temps réel

- Temps réel en général
  - Définition
  - Tableau comparatif
  - Temps réel mou
  - Les limites du temps réel mou
- Outils de mesure des performances
- Exemples
- Conclusion
- Bibliographie

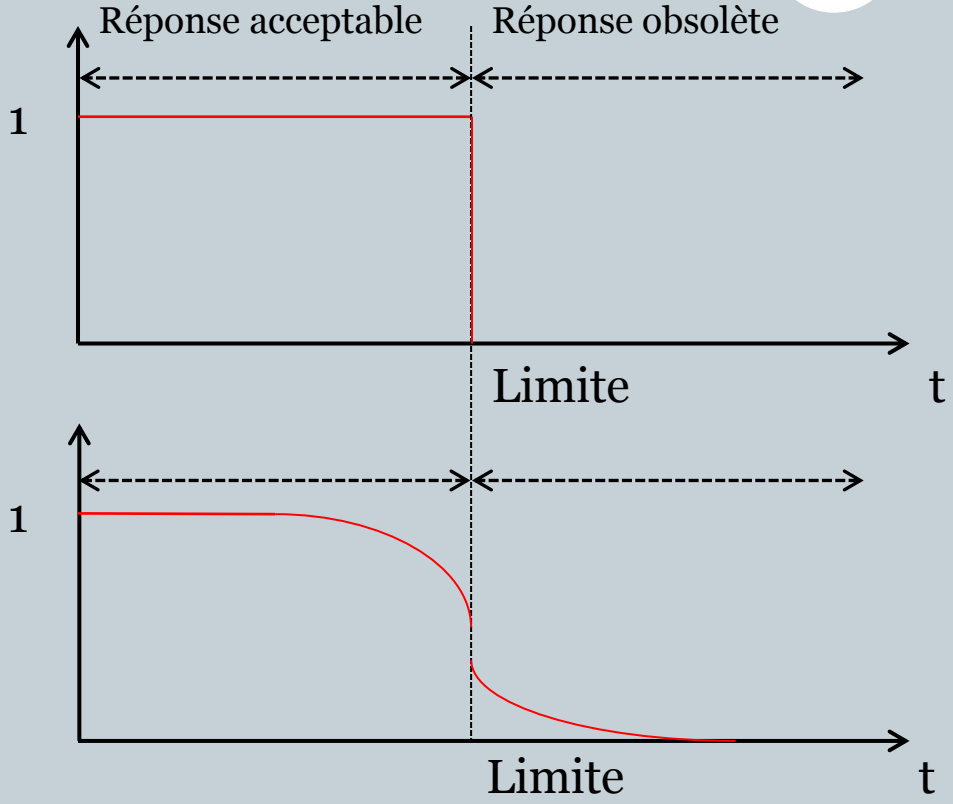
Plan : 3ème partie – Temps réel

- **Temps réel en général**
  - **Définition**
  - **Tableau comparatif**
  - **Temps réel mou**
  - **Les limites du temps réel mou**
- Outils de mesure des performances
- Exemples
- Conclusion
- Bibliographie

Définition 1/2

- Définition : un système est dit temps réel lorsque l'information après acquisition et traitement reste encore pertinente
  - Temps réel mou (système acceptant des variations de l'ordre de la milliseconde dans le traitement de données) : répartition égalitaire du temps processeur. Ces systèmes garantissent un temps **moyen** d'exécution pour chaque tâche
  - Temps réel dur : gestion stricte du temps est nécessaire pour conserver l'intégrité du service rendu. Ces systèmes garantissent un temps **maximum** d'exécution pour chaque tâche

Définition 2/2



Temps réel dur

Temps réel souple

Tableau comparatif

	Timer		Réponse aux interruptions externes		Priorités
	Périodes minimales	Fluctuations maximales	Temps de réponse moyen	Fluctuation des temps de réponse	
Linux « vanilla »	QQ 10µs	QQ 100 µs	QQ µs	QQ 100 µs	Les tâches applicatives les plus prioritaires sont néanmoins préemptées par des traitements d'interruptions éventuellement longs.
Linux -RT	QQ 10µs	QQ 10 µs	QQ µs	QQ 10 µs	Certaines tâches applicatives peuvent avoir une priorité supérieure à celles des traitements d'interruptions.
Xenomai	QQ 10µs	QQ µs	QQ µs	QQ µs	Tâches Xenomai plus prioritaires que celles du noyau natif

Temps réel mou 1/5

- Utilitaire chrt pour lancer un processus en temps réel même si celui-ci n'a pas été prévu
- Exemple:
  - chrt –f 50 ./executable //executable lancé en ord. Fifo de priorité 50
  - chrt –r 10 ./executable //executable lancé en ord. RR de priorité 10
  - chrt –o 0 ./executable //executable lancé en ord. partagé
  - chrt –p 12345 //affiche l'ord. Et la priorité du process 12345
  - chrt –pf 50 12345 //passe le process 12345 en FIFO de prio 50
  - chrt –pr 30 12345 //passe le process 12345 en RR de prio 30

Temps réel mou 2/5

- Utilitaire taskset qui permet de fixer ou de récupérer l'affinité d'un processus au CPU en console.
- Exemple :
  - taskset –c 1 ./executable //processus lancé sur le CPU1
  - taskset 0,1 ./executable //processus lancé sur le CPU 0 et 1
  - taskset –pc 0 2013 //migration du processus 2013 sur le CPU 0
  - taskset –p 2013 //affiche l'affinité du processus 2013

- En programmation : utilisation 2 fonctions :

```
#define _GNU_SOURCE  //constante symbolique => pas POSIX
#include <sched.h>

int sched_setaffinity(pid_t                pid,
                      size_t          cpusetsize,
                      cpu_set_t          *cpuset);

int sched_getaffinity(pid_t                pid,
                      size_t          cpusetsize,
                      cpu_set_t          *cpuset);
```

Identifiant du processus

Taille du type de donnée

Usage du CPU

Temps réel mou 3/5

- Echelle des priorités:
  - Tâche temps réel située dans l'échelle de priorité [1,99]
  - Tâche de priorité 0 : temps partagé
- La tâche de priorité 49 ne s'exécutera que lorsque de la tâche de priorité 50 s'endorme.
- Si une tâche de priorité 51 se réveille, la tâche 50 est préemptée !

Temps réel mou 4/5

- Quand 2 tâches ont le même niveau de priorité ?
- 2 types de traitement suivant l'ordonnancement temps réel utilisé :
  - FIFO (First In, First Out) : pas de préemption de la tâche en cours si la nouvelle est de même priorité
  - Round Robin (tourniquet) : préemption de la tâche en cours au bout d'un moment par une tâche de même niveau de priorité

Temps réel mou 5/5

- Il existe un garde-fou temps réel natif depuis le noyau 2.6.25 de Linux. Une petite partie du temps CPU ≈5% est réservé aux tâches temps partagé même quand les tâches temps réels s'éxecutent
- Temps réel souple adapté pour faire fonctionner des tâches périodiques de quelques centaines de microsecondes.

Les limites du temps réel mou

- Problème temps réel (mou) classiques
  - Un thread créé préempte son créateur et l'empêche de créer les autres threads équivalent => mise en place de barrières POSIX
  - Inversion de priorité => mise en place du Priority Inheritance Protocol (PIP) activable à l'initialisation du mutex
  - Un thread lâchant un mutex et le demandant à nouveau au détriment des autres => employez la fonction **sched_yield( )**

Plan : 3ème partie – Temps réel

- Temps réel en général
  - Définition
  - Tableau comparatif
  - Temps réel mou
  - Les limites du temps réel mou
- **Outils de mesure des performances**
- Exemples
- Conclusion
- Bibliographie

Outils de mesure des performances

- Outils de test de système temps réel
  - Cyclitest : mesure les fluctuations et les latences de divers paramètres
  - Hwlatency: détecte les latences dues au matériel
  - Hackbench: mesure la commutation entre threads et processus
  - Pi_stress : test l'implémentation des héritages de priorités sur les mutex
  - Et bien d'autres sur http://rt.wiki.kernel.org

Plan : 3ème partie – Temps réel

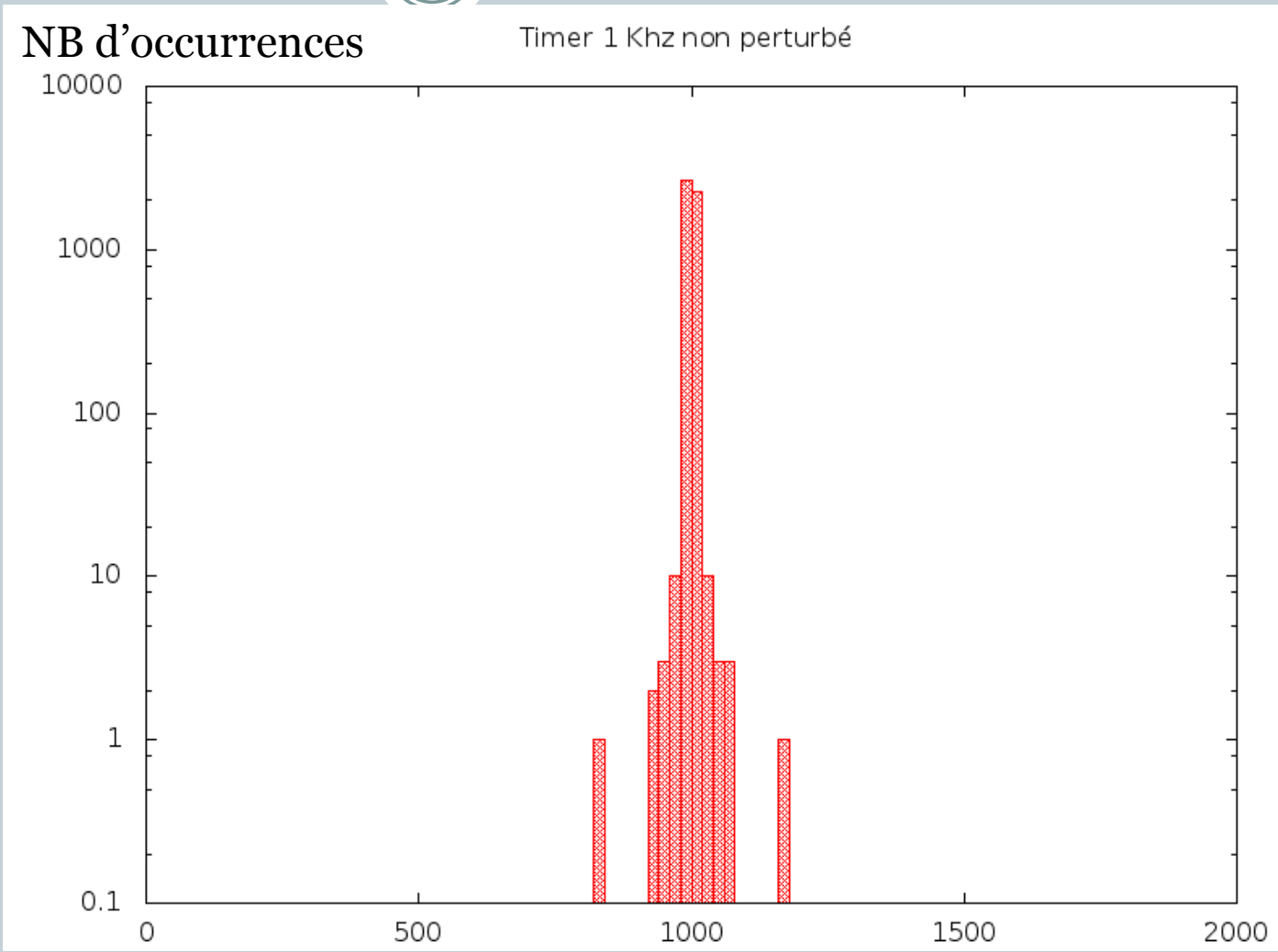
- Temps réel en général
  - Définition
  - Tableau comparatif
  - Temps réel mou
  - Les limites du temps réel mou
- Outils de mesure des performances
- **Exemples**
- Conclusion
- Bibliographie

Exemple : temps partagé

Durées entre déclenchements successif du timer

- Mesures importantes autour de la valeur de consigne : 1ms

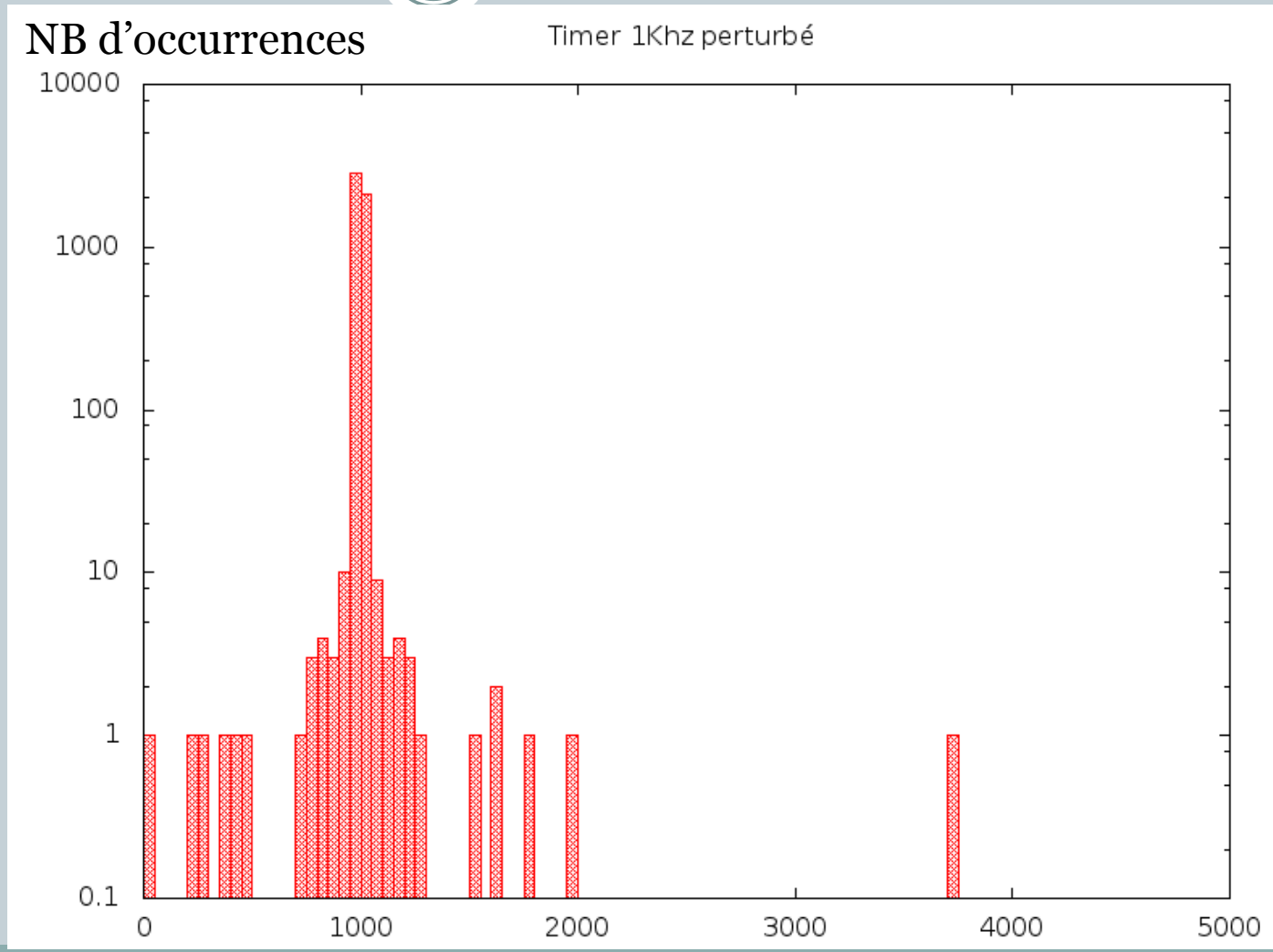
- Quelques écart existent autour de la valeur



Exemple : temps partagé

Durées entre déclenchements successif du timer

- Dispersion plus large autour de la valeur de consigne 1ms
- Apparition de valeurs extrême à 3,7ms

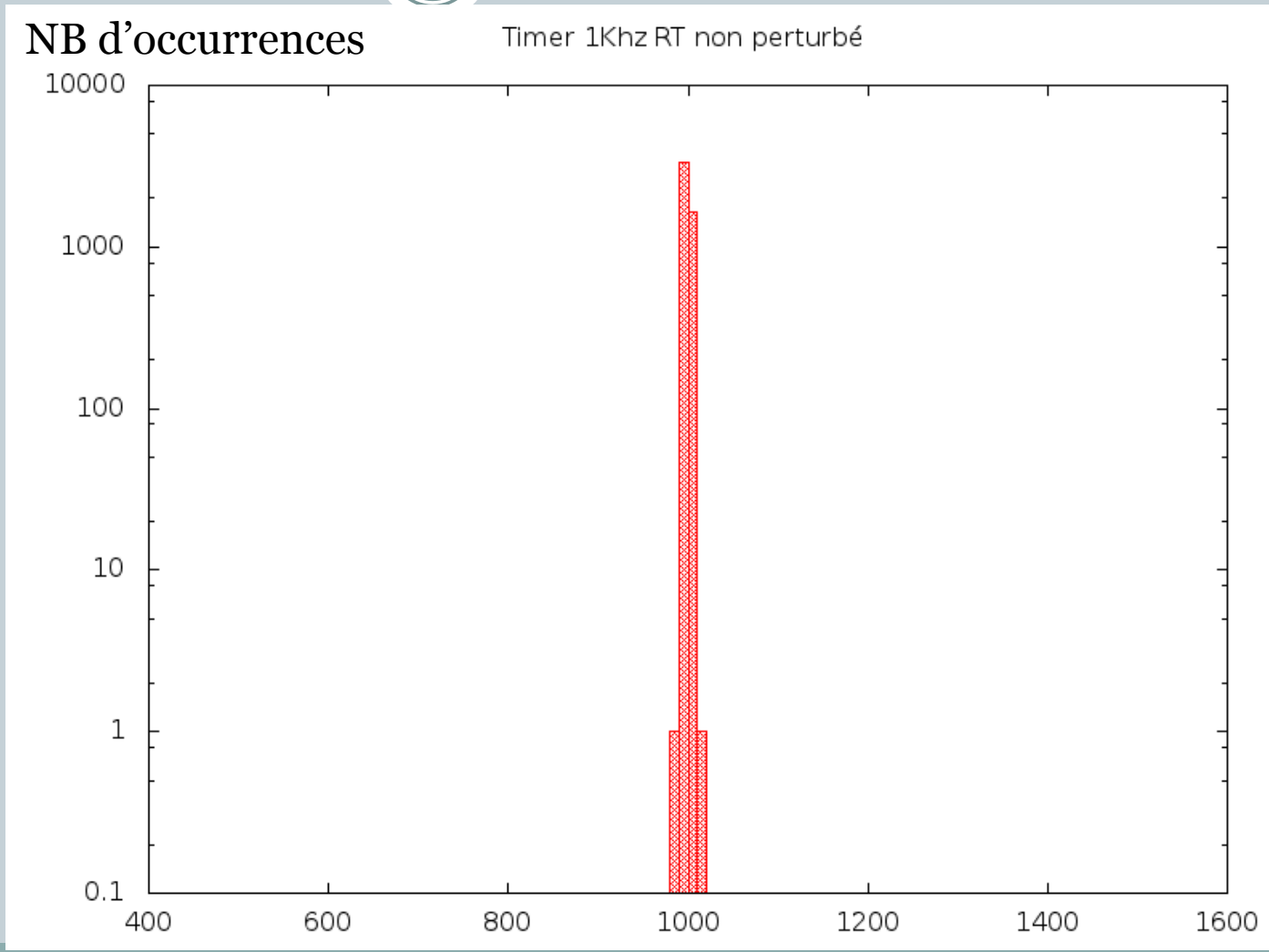


Exemple : temps réel-patch Linux-rt

Durées entre déclenchements successif du timer

- Taux de mesures importantes autour de la valeur de consigne : 1ms

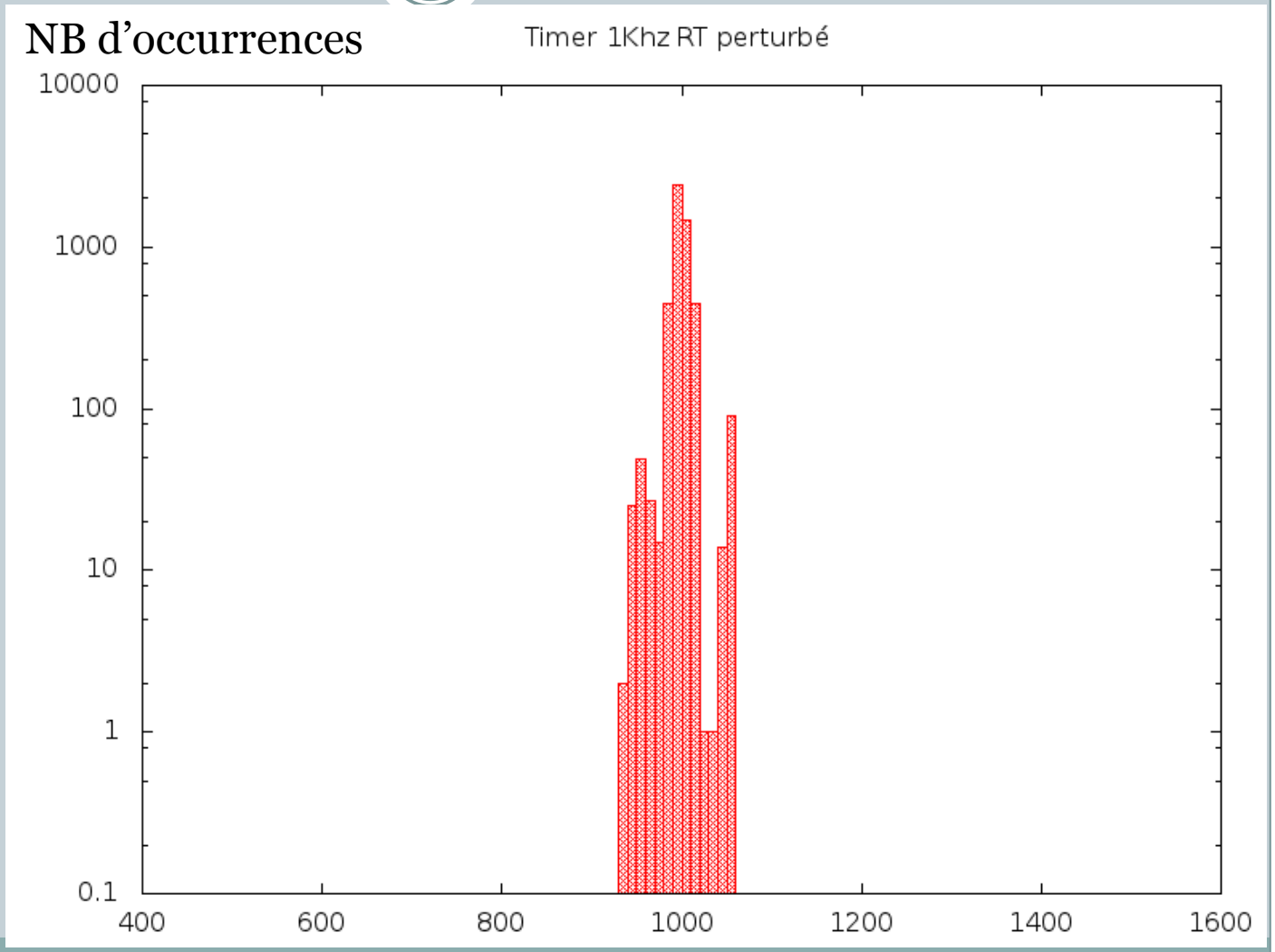
- Aucun écart autour de la valeur !!



Exemple : temps réel-patch Linux-rt

Durées entre déclenchements successif du timer

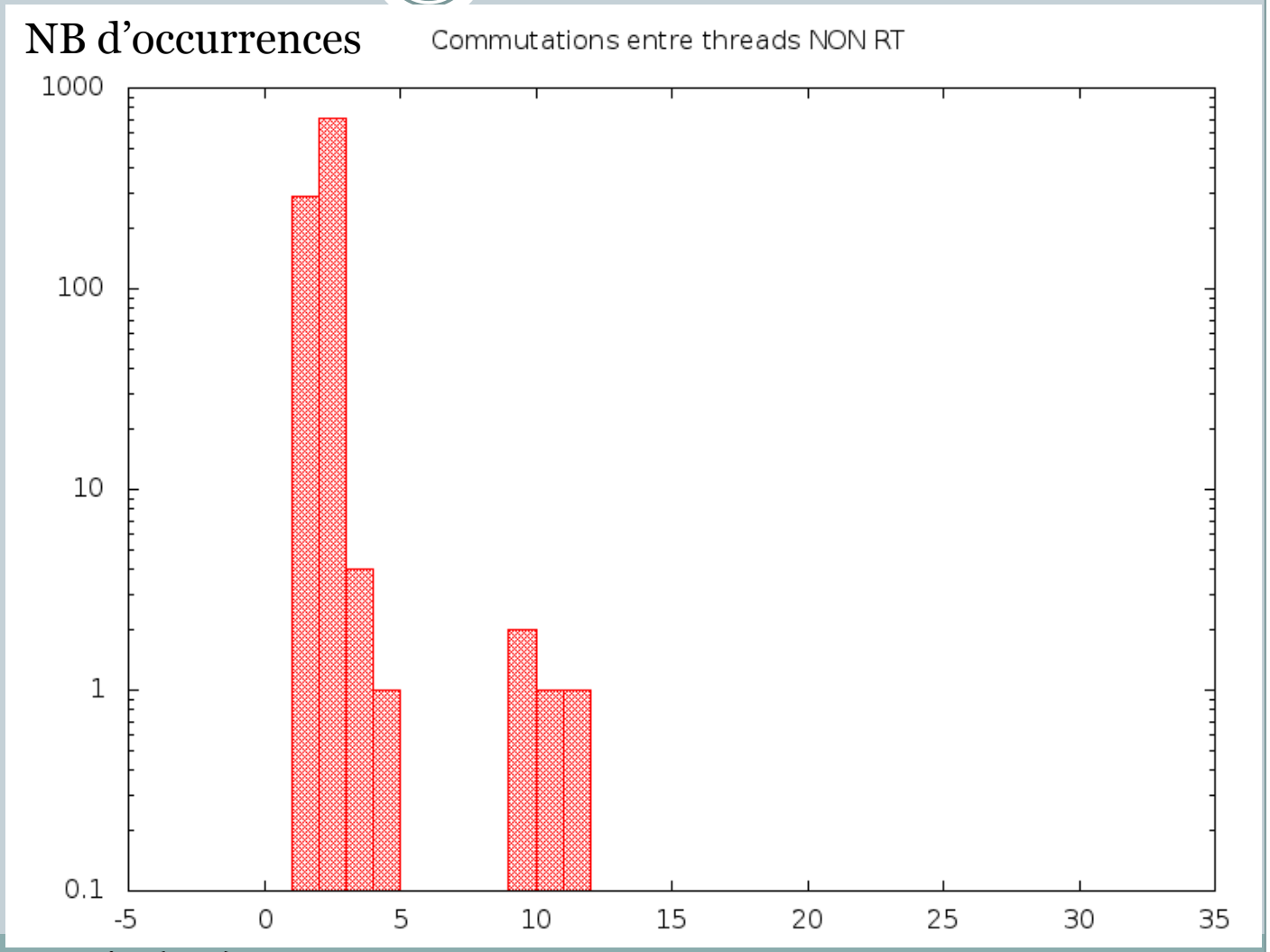
- Petites dispersions autour de la valeur de consigne 1ms
- Aucune valeur extrême !!!



Exemple : temps partagé

Temps de commutations entre 2 threads synchronisés sur 1 ressource partagée (mutex)

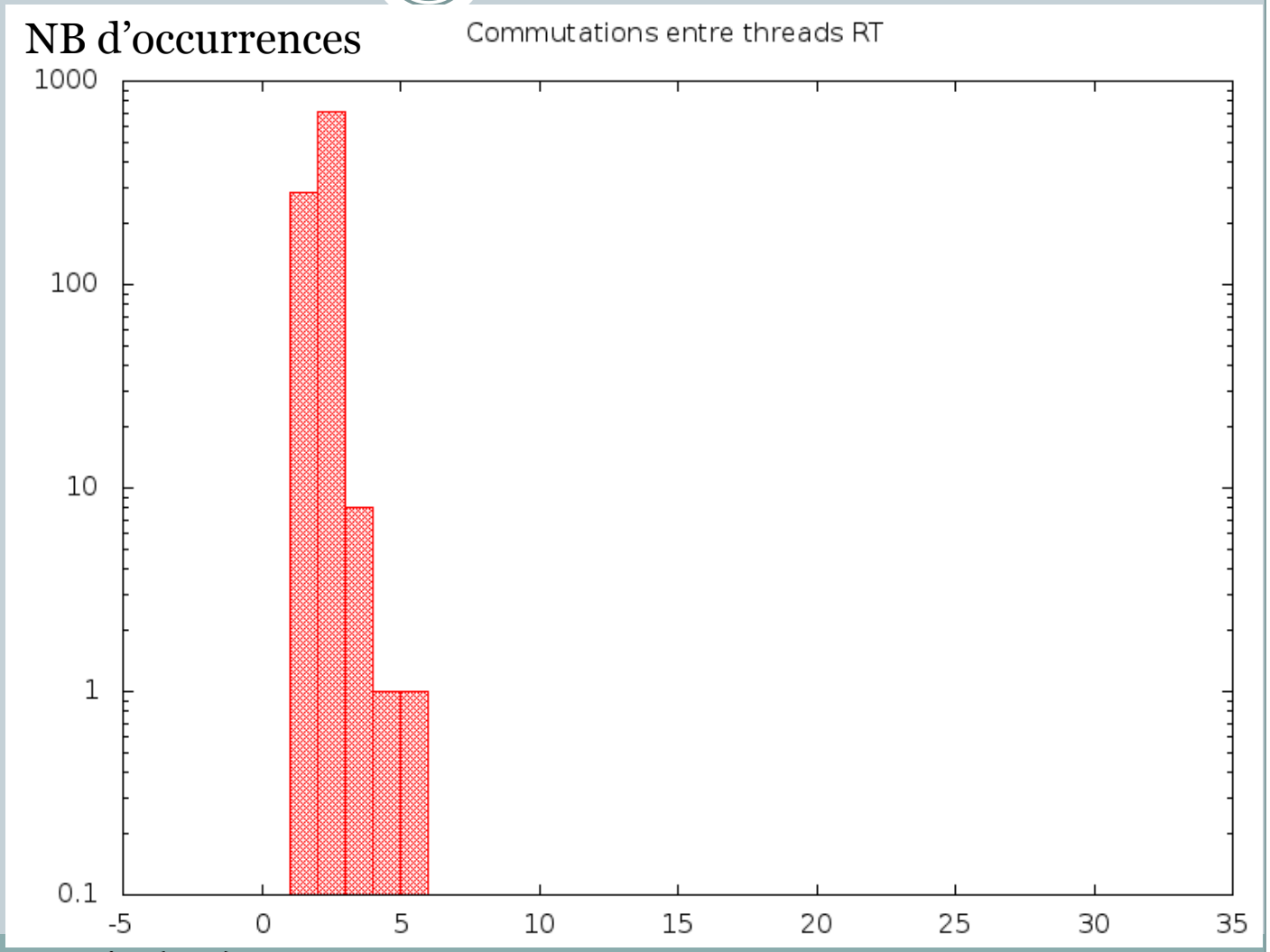
- Petites dispersions
- Quelques valeurs extrêmes !!!



Exemple : temps réel-patch Linux-rt

Temps de commutations entre 2 threads synchronisés sur 1 ressources partagée (mutex)

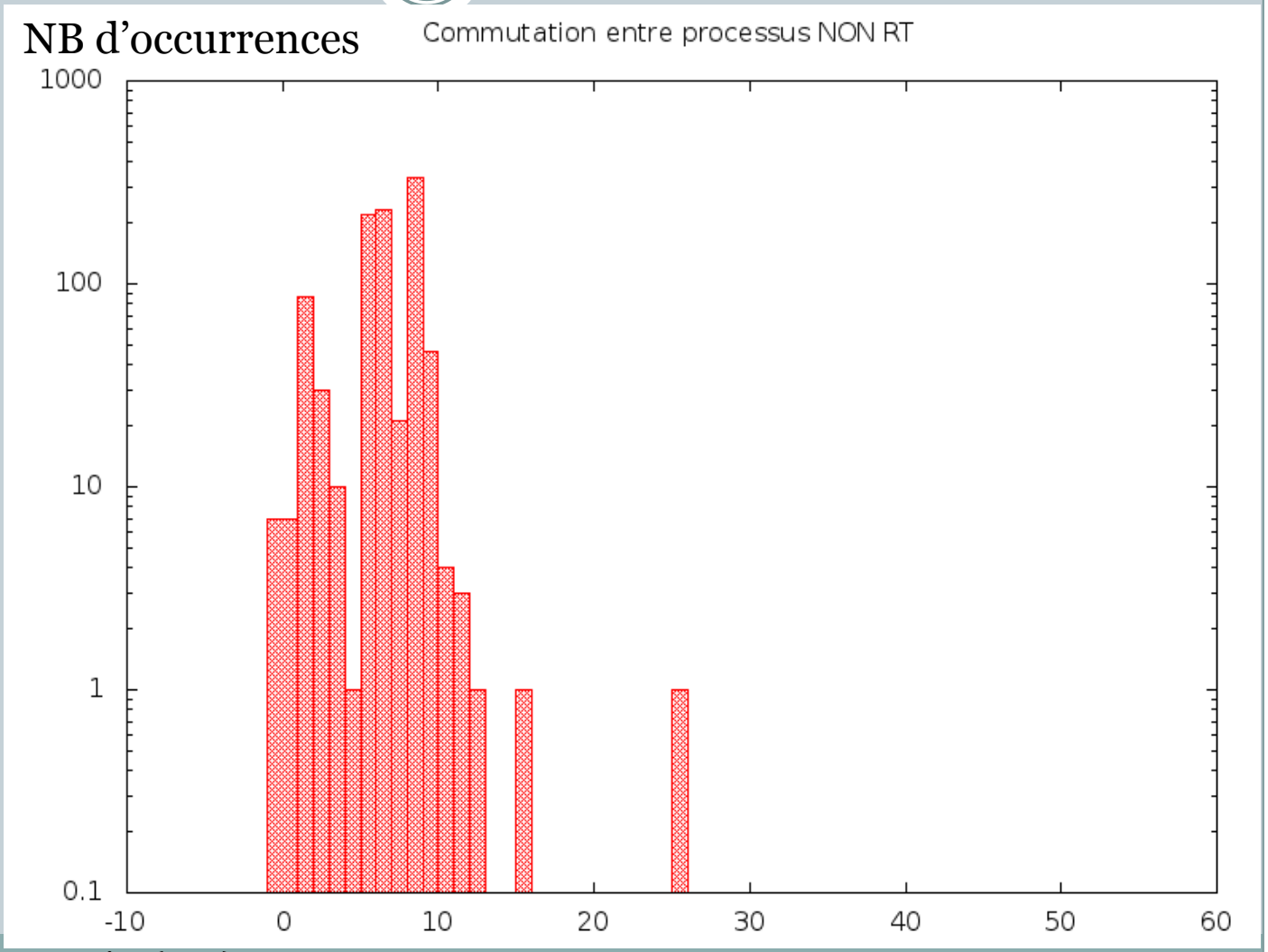
- Petites dispersions autour de la valeur
- Aucune valeur extrême !!!



Exemple : temps partagé

Temps de commutations entre 2 processus synchronisés sur 1 ressource partagée (mutex)

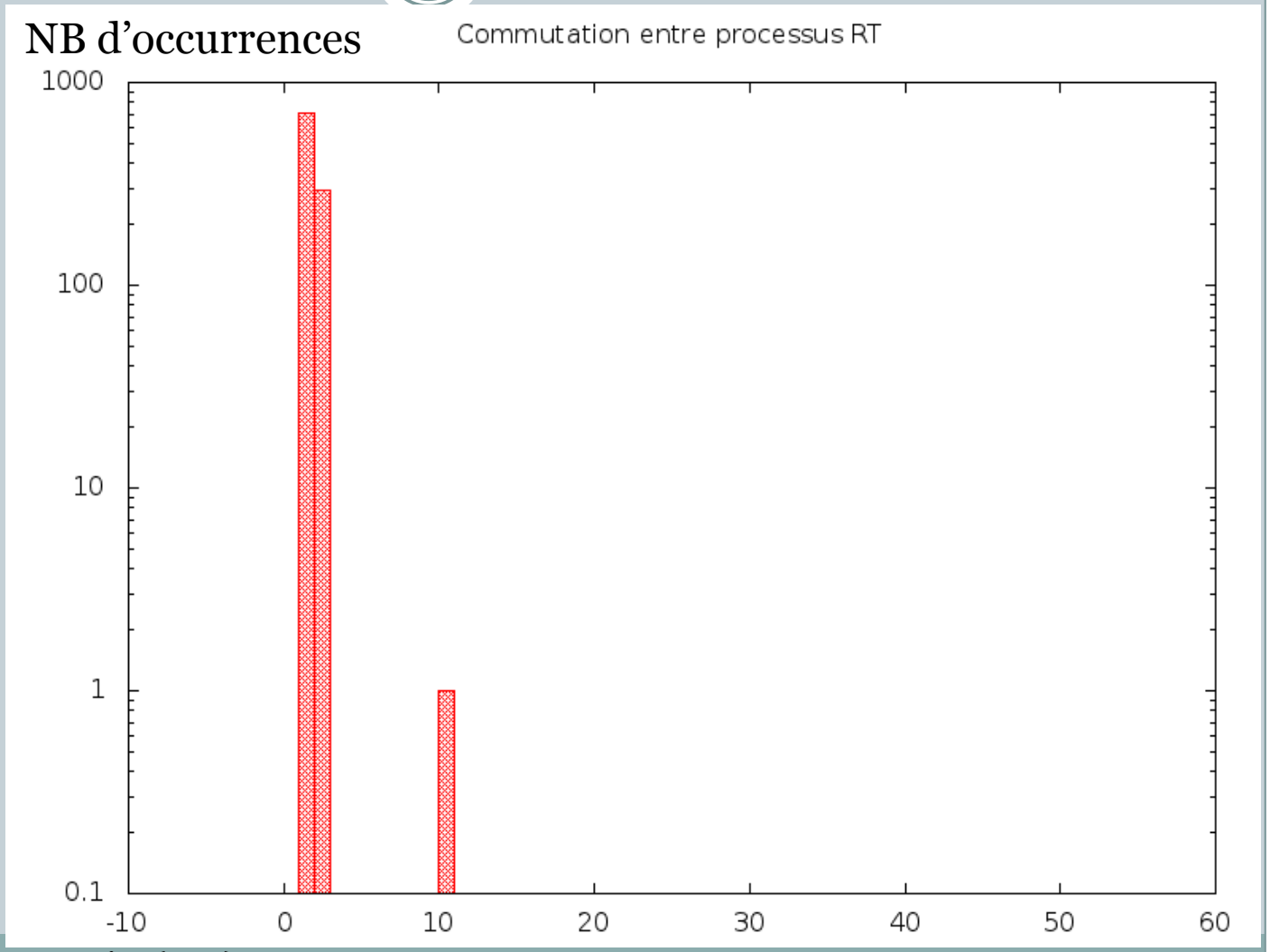
- Les durées moyennes et maximales sont plus longues
- Beaucoup plus de dispersement que les threads



Exemple : temps réel-patch Linux-rt

Temps de commutations entre 2 processus synchronisés sur 1 ressource partagée (mutex)

- Petites dispersions autour de la valeur de consigne quelques millisecondes
- 1 valeur extrême !!!



Plan : 3ème partie – Temps réel

- Temps réel en général
  - Définition
  - Tableau comparatif
  - Temps réel mou
  - Les limites du temps réel mou
- Outils de mesure des performances
- Exemples
- **Conclusion**
- Bibliographie

Conclusion

- Un système en temps partagé classique répond à beaucoup de critère
- Le temps réel « mou » (patch Linux-RT) est facilement adaptable pour de bon résultats sans connaissances particulières supplémentaires(même API)
- Le temps rée « dur » est en fait pas si utilisé que cela. Seulement 10% des projets en industrie sont concernés !

Plan : 3ème partie – Temps réel

- Temps réel en général
  - Définition
  - Tableau comparatif
  - Temps réel mou
  - Les limites du temps réel mou
- Outils de mesure des performances
- Exemples
- Conclusion
- **Bibliographie**

Bibliographie 1/3

- Les sites web :
  - http://www.blup.fr/programmation-systeme-en-c-sous-linux
  - http://www.advancedlinuxprogramming-fr.org
  - http://www.blaess.fr/christophe/
  - http://www.kernel.org
  - http://ltp.sourceforge.net
  - http://rt.wiki.kernel.org
  - http://www.xenomai.org
  - http://www.rtail.org
  - http://www.osadl.org
  - http://www-igm.univ-mlv.fr/~paumier/enseignement.html

Bibliographie 2/3

- Les magazines :

Bibliographie 3/3

- Salon RTS (mars/avril) Paris porte de Versailles (vous pouvez demander les présentations aux séminaristes :

http://www.salons-solutions electroniques.com/

- Les livres: