


Plan du cours

-  Taxonomie des systèmes informatiques
-  Systèmes temps réel
-  Spécificités des OS pour le temps réel
-  **L'OS Xenomai pour le temps réel**
-  Systèmes embarqués
-  Linux pour l'embarqué
-  Marché des OS pour le temps réel et l'embarqué
-  Modélisation d'applications temps réel avec UML 2.x

La solution Xenomai

- Origine et historique
- Présentation
- Architectures single-kernel et dual-kernel
- Services Xenomai
- Programmation sous Xenomai



Origine et historique

- **Projet indépendant fondé en 2001**

- Xenomai v0.5 – Septembre 2001
- Xenomai v1.1.1 – Décembre 2002

- **Intégration au projet RTAI en 2003**

- RTAI/fusion v0.1 – Juin 2004

- **Indépendance reprise en 2005**

- Xenomai v2.0 – Octobre 2005 (base RTAI/fusion v0.9.1)
- Xenomai v2.1 – Mars 2006
- Xenomai v2.2 – Juillet 2006
- Xenomai v2.3 – Décembre 2006
- Xenomai v2.4 – Décembre 2007
- Xenomai v2.5 – Janvier 2010
- Xenomai v2.6 – Novembre 2011
- **Xenomai v3.0 – Octobre 2014**

Xenomai (1)



- Xenomai est un **système d'exploitation temps réel (RTOS)**
- Garanties temps réel **dur** et/ou **soft** (*selon le type de configuration à l'install*)
- RTOS multi plate-formes + faible latence
- **Emulation de RTOS traditionnels** (VxWorks, pSOS, ...)
- Licence GPL v2 (cœur), **LGPL v2.1** (interfaces)

Xenomai (2)



Cibles supportées :

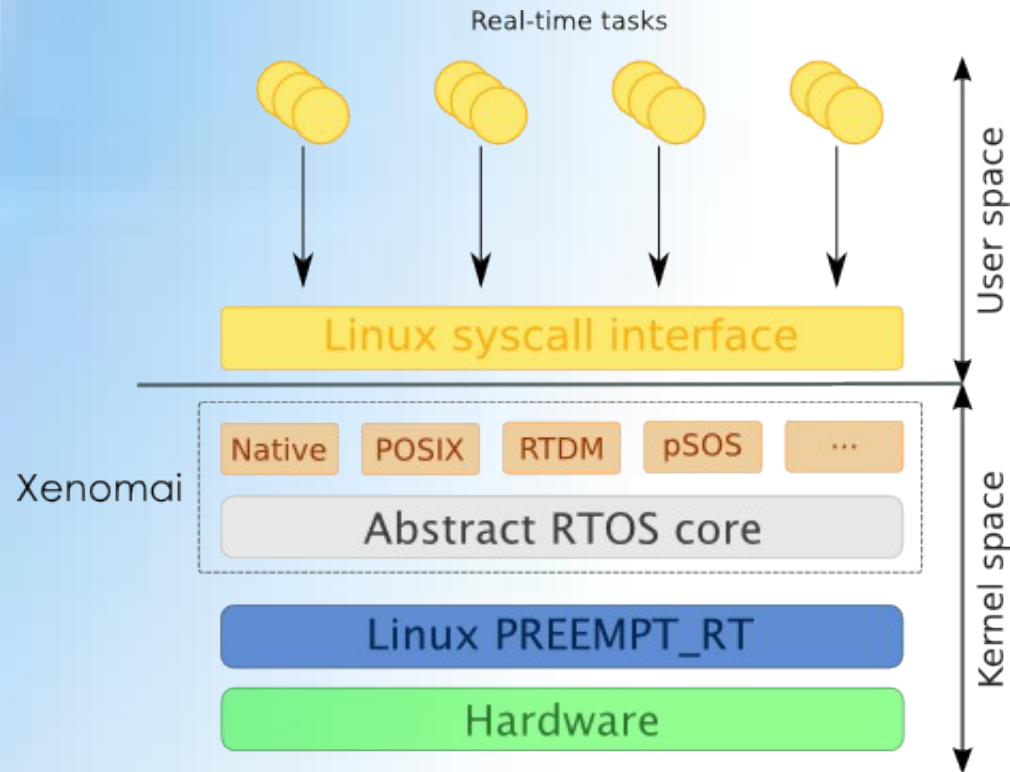
- ARM, Atmel, Freescale, Blackfin, Samsung, STMicroelectronics, Broadcom, Texas Instruments, Linksys/Cisco, Raspberry Pi...
- I386, ia64 (Intel Itanium), Power PC
- x86_64
 - Intel : Pentium 4, Pentium D, Pentium Extreme Edition, Celeron D, Xeon, Pentium Dual-Core
 - AMD : Athlon 64, Athlon 64 FX, Athlon 64 X2, Turion 64, Turion 64 X2, Opteron, Sempron

Xenomai (3)

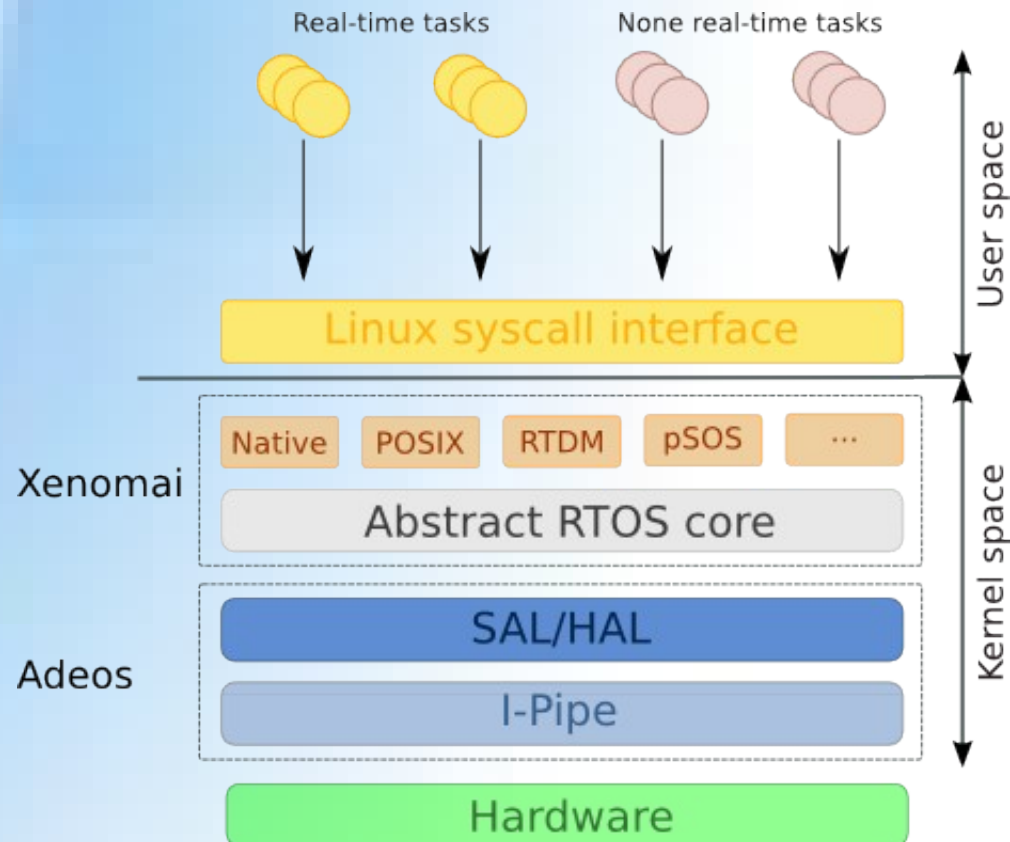


- 2 types d'installation
- Configuration single-kernel
 - Extension appelée '**mercury**'
 - Temps réel soft
- Configuration dual-kernel
 - Extension appelée '**cobalt**'
 - Temps réel hard

Configuration single-kernel

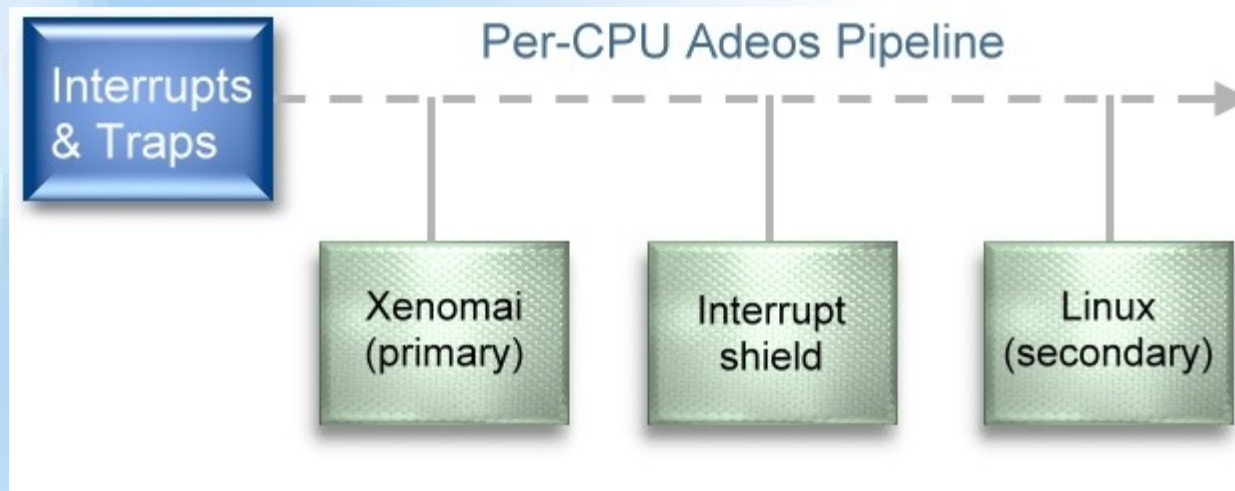


Configuration dual-kernel (1)



Configuration dual-kernel (2)

- Comment faire cohabiter les 2 OS sur un même matériel ?
- **Solution** : intercaler une couche (I-PIPE ADEOS) entre le matériel et les OS



Configuration dual-kernel (3)

- Les domaines d'ordonnancement

- Domaine Xenomai → déterministe
- Domaine Linux → non déterministe

- Les modes d'exécution d'une tâche temps réel

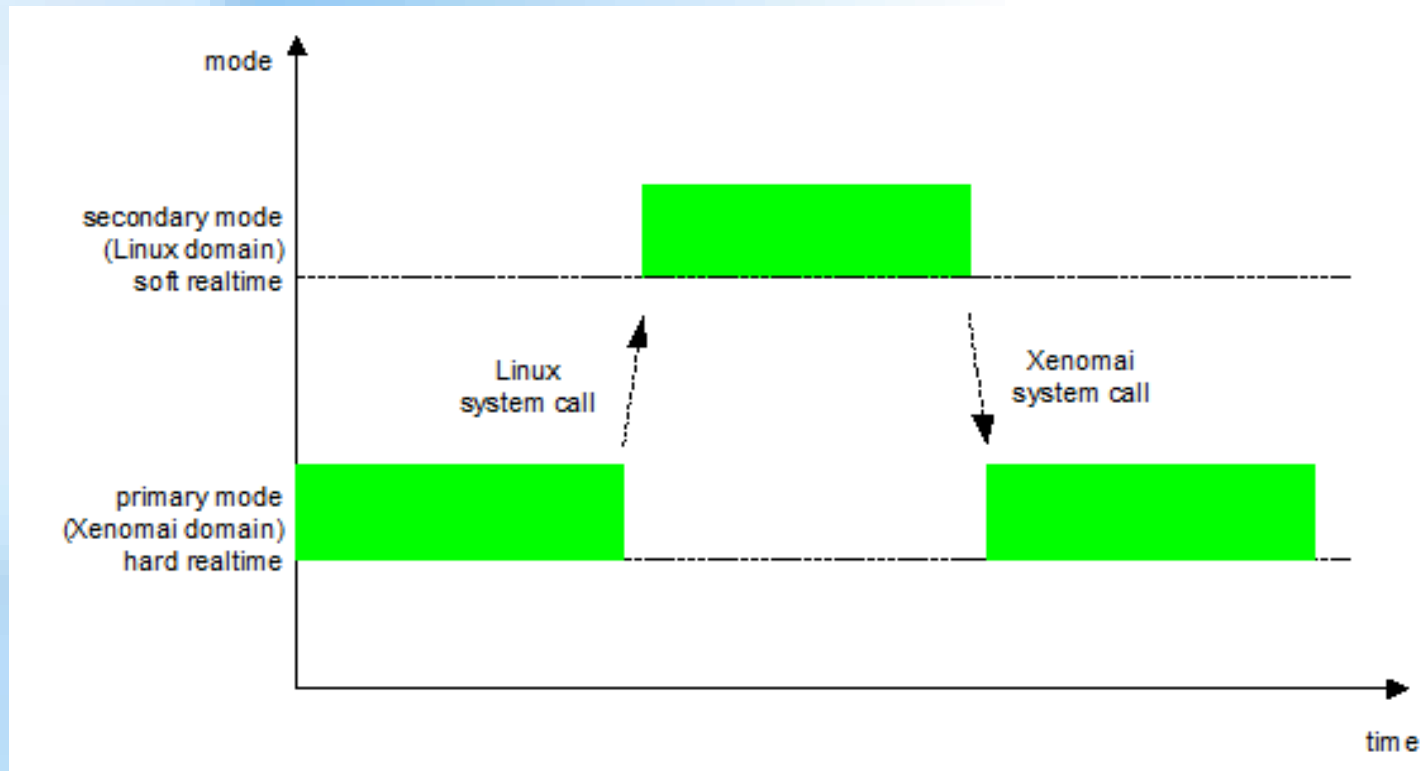
- Mode primaire → exécution TR dur au sein du domaine Xenomai
- Mode secondaire → exécution TR mou (exécution bornée) au sein du domaine Linux

- Une tâche temps réel migre **automatiquement** d'un domaine à l'autre *en fonction des appels système qu'elle réalise*

- Sa priorité reste constante quel que soit le domaine dans lequel elle se situe

Configuration dual-kernel (4)

- Migration interdomaines d'une tâche temps réel



Services Xenomai



- **Supporte :**

- ordonnancement à priorités fixes (256 niveaux)
- ordonnancement sans priorités : round-robin
- *synchronisation* (sémaphores, mutex)
- gestion des ressources partagées : PIP
- *communication* (files de messages, tubes)
- gestion des interruptions
- allocation dynamique de mémoire spécifique temps réel
- watchdogs
- timers

La programmation sous Xenomai



API native vs. API POSIX

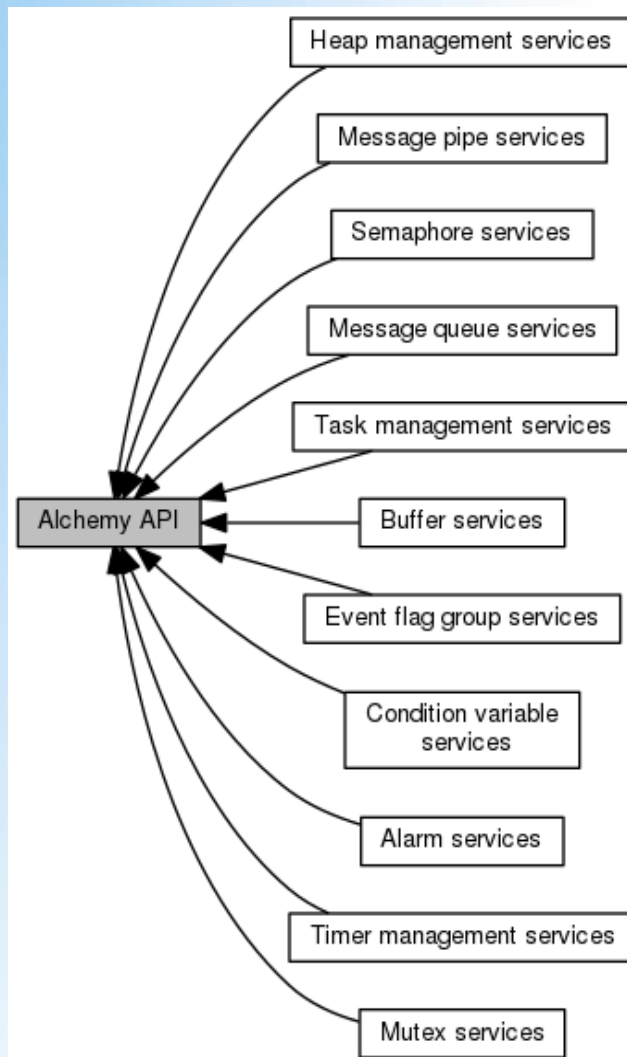
Organisation structurelle d'une application Xenomai

Programmation de tâches temps-réel

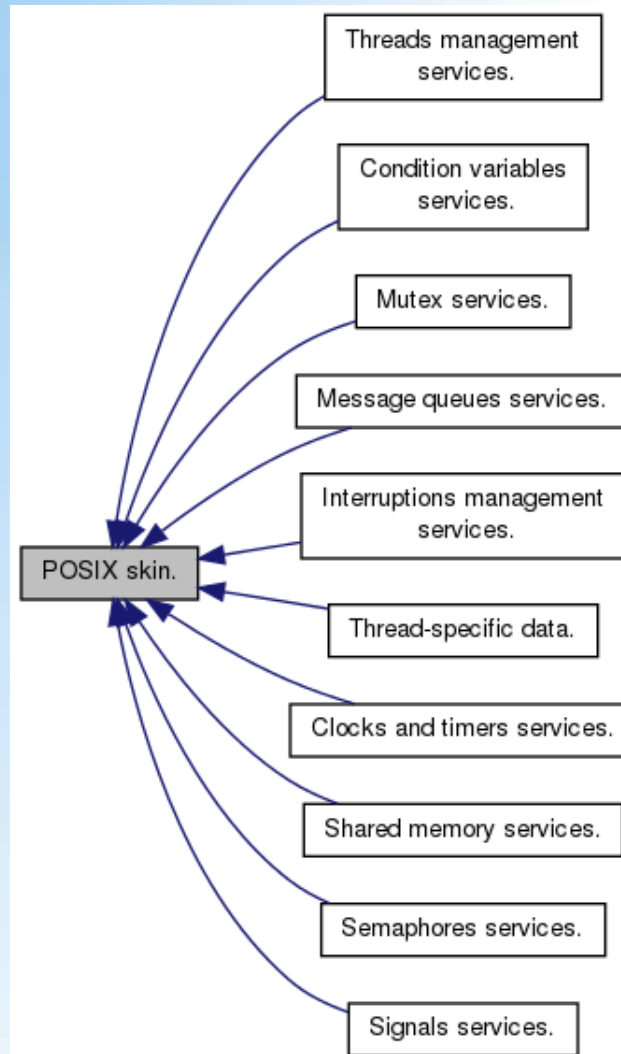
Mécanismes IPC natifs

Compilation et lancement une application Xenomai

API native (Alchemy)



API POSIX



Organisation structurelle d'une application Xenomai (1)

- Inclusions de bibliothèques

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <mqueue.h>
. . .
```

Librairies **POSIX**

Organisation structurelle d'une application Xenomai (2)

- Définitions générales

```
#define STACK_SIZE_IN_K0 2000
#define PRIORITY_T1 2
#define PRIORITY_T2 4
```

Caractéristiques
des tâches temps-réel

```
#define PERIOD_IN_MICROS 500000
. . .
```

Caractéristiques
de l'ordonnancement

Organisation structurelle d'une application Xenomai (3)

- Déclaration des threads

```
pthread_t TacheHorloge;  
pthread_t TacheCapteur;  
pthread_t TacheActionneur;  
pthread_t TacheControle;  
...
```

- Déclaration des attributs des threads

```
pthread_attr_t TacheHorlogeAttributes;  
pthread_attr_t TacheCapteurAttributes;  
pthread_attr_t TacheActionneurAttributes;  
pthread_attr_t TacheControleAttributes;  
...
```

Organisation structurelle d'une application Xenomai (2)

- Définitions générales

```
#define STACK_SIZE_IN_K0 2000
#define PRIORITY_T1 2
#define PRIORITY_T2 4
```

Caractéristiques
des tâches temps-réel

```
#define PERIOD_IN_MICROS 500000
. . .
```

Caractéristiques
de l'ordonnancement

Organisation structurelle d'une application Xenomai (4)

- **Déclaration des mécanismes IPC**

```
sem_t synchroLevel1;  
sem_t synchroLevel2;  
...  
pthread_mutex_t mutex1;  
pthread_mutex_t mutex2;  
...  
mqd_t msgq1;  
...
```

Organisation structurelle d'une application Xenomai (5)

- **Code des fonctions des différentes tâches temps réel**

```
static void CodeTacheCapteur(void *arg)
{
    int n=0;
    printf("Tache Capteur en execution...\n");
    . . .
};

static void CodeTacheActionneur(void *arg) {
    int i=0;
    for (i=0; i<10000; i++)
        . . .
    . . .
}
```

Organisation structurelle d'une application Xenomai (6)

• Fonction principale de lancement et d'arrêt

```
int main(void){
mlockall( MCL_CURRENT | MCL_FUTURE );

...

if ((n=createPosixTask("TacheCapteur", PRIORITY_TACHE_CAPTEUR,
STACK_SIZE_IN_KO, PERIOD_IN_MICROS, &TacheCapteur,
&TacheCapteurAttributes, CodeTacheCapteur))!=0)
{   printf("Init task error %d\n, n);
    return 0;
}

...

pthread_join(&TacheCapteur);
return 0;
}
```

Les threads POSIX (1)

- Fonctions de gestion des threads :

pthread_create() : crée et initialise un nouveau thread

pthread_exit() : arrêt d'un thread

pthread_join() : attente de la terminaison d'un thread

pthread_setname_np() : nommage d'un thread

Les threads POSIX

- Fonctions de gestion des attributs des threads :

pthread_attr_init() : crée et initialise par défaut un ensemble d'attributs

pthread_attr_setdetachstate() : arrêt d'un thread

pthread_attr_setschedpolicy() : définit la politique d'ordonnancement

pthread_attr_setschedparam() : définit un ensemble de paramètres
(dont la priorité)

pthread_attr_setstacksize() : définit la taille de pile

La gestion du timer

- Initialisation et réglage du timer :

timerfd_create() : création d'un timer

timerfd_settime() : réglage de la période d'activation du timer

- Récupération d'informations temporelles

clock_gettime() : retourne la valeur courante du timer en ns

Mécanismes IPC POSIX

- Les sémaphores d'exclusion mutuelle
- Les sémaphores de synchronisation
- Les files de messages



Les sémaphores d'exclusion mutuelle

- Fonctions de gestion des mutexes :

pthread_mutex_init() : crée et initialise un nouveau mutex

pthread_mutex_destroy() : détruit un mutex

pthread_mutex_lock() : opération d'acquisition du mutex (*.P()*)

pthread_mutex_unlock() : opération de libération du mutex (*.V()*)

- Version temporisée :

pthread_mutex_timedlock() : opération d'attente du mutex (*.P()*)
jusqu'à une date absolue

Les sémaphores de synchronisation

- Fonctions de gestion des sémaphores compteurs :

sem_init() : crée et initialise un nouveau sémaphore

sem_destroy() : détruit un sémaphore

sem_wait() : opération d'attente du sémaphore (**.P()**)

sem_post() : opération de signal du sémaphore (**.V()**)

- Version temporisée :

sem_timedwait() : opération d'attente du sémaphore (**.P()**)

jusqu'à une date absolue

Les files de messages

- Fonctions de gestion des files de messages :

mq_open() : crée et ouvre un accès une nouvelle file

mq_receive() : lit une donnée dans la file

mq_send() : écrit une donnée dans la file

mq_close() : ferme l'accès à une file

Compilation d'une application Xenomai

- Utilisant l'API POSIX

```
gcc -o appli appli.c `xeno-config --skin posix -cflags`  
-lpthread -lrt
```

- Utilisant l'API native

```
gcc -o appli appli.c `xeno-config --skin alchemy -cflags`  
-lpthread -lrt
```