Realtime applications with RTAI

R.KOCIK – Embedded Systems department

Outline

I. Which RTOS to choose ?

II. Why not to choose Linux ?

III. Realtime Linux

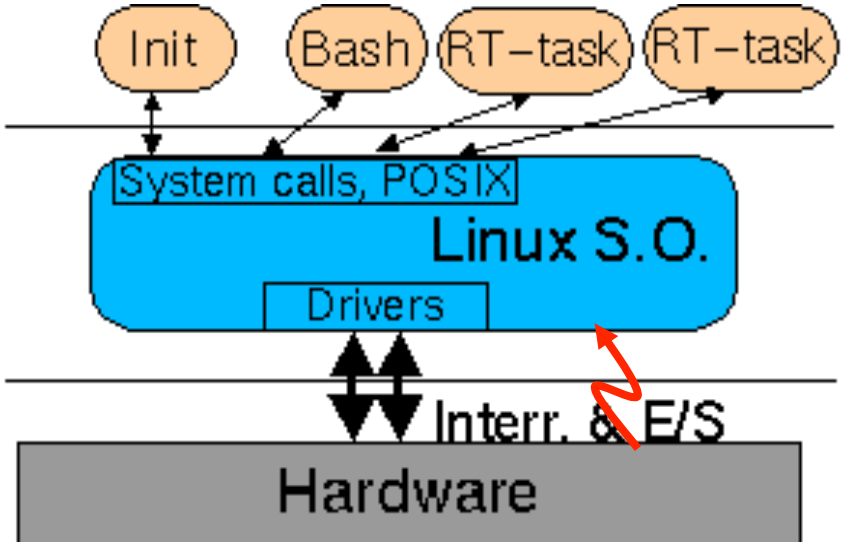
IV. Using RTAI

V. RT TASK scheduling with RTAI

VI. Synchronisation, Communication

VII. RTAI: conclusion

VIII. And other RTOS ….



I. WHICH RTOS TO CHOOSE?

Specification of a RTOS

- Minimum to manage realtime
  - Interrupt management,
  - Time base,
  - Tasks declaration, creation, execution, preemption, destruction
- Minimum cost (speed, memory,…)

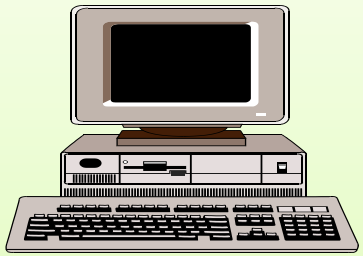
Additionnal Services

- Memory Management
- Communications with devices (driver)
- Tools for communication an synchronisation
- Human Machine Interface

Development Architecture

- Cross Development

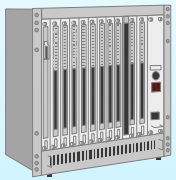
Development computer

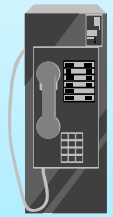


- Source writing
- Cross compilation
- Linking :
  - -code
  - -RT kernel and librairies
  - -loader

- Simulation
- Download
- Debug

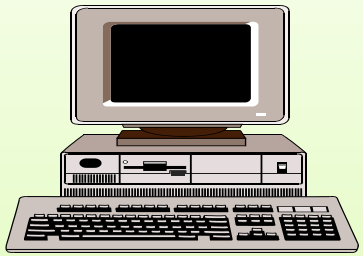
Target computer





- Embedded system
- I/O
- Controlled system
- Sometimes keyboard/Screen

Development Architecture II



Development & target computer

- Source writing
- Cross compilation
- Linking :
  - -code
  - -RT kernel and librairies
  - -loader

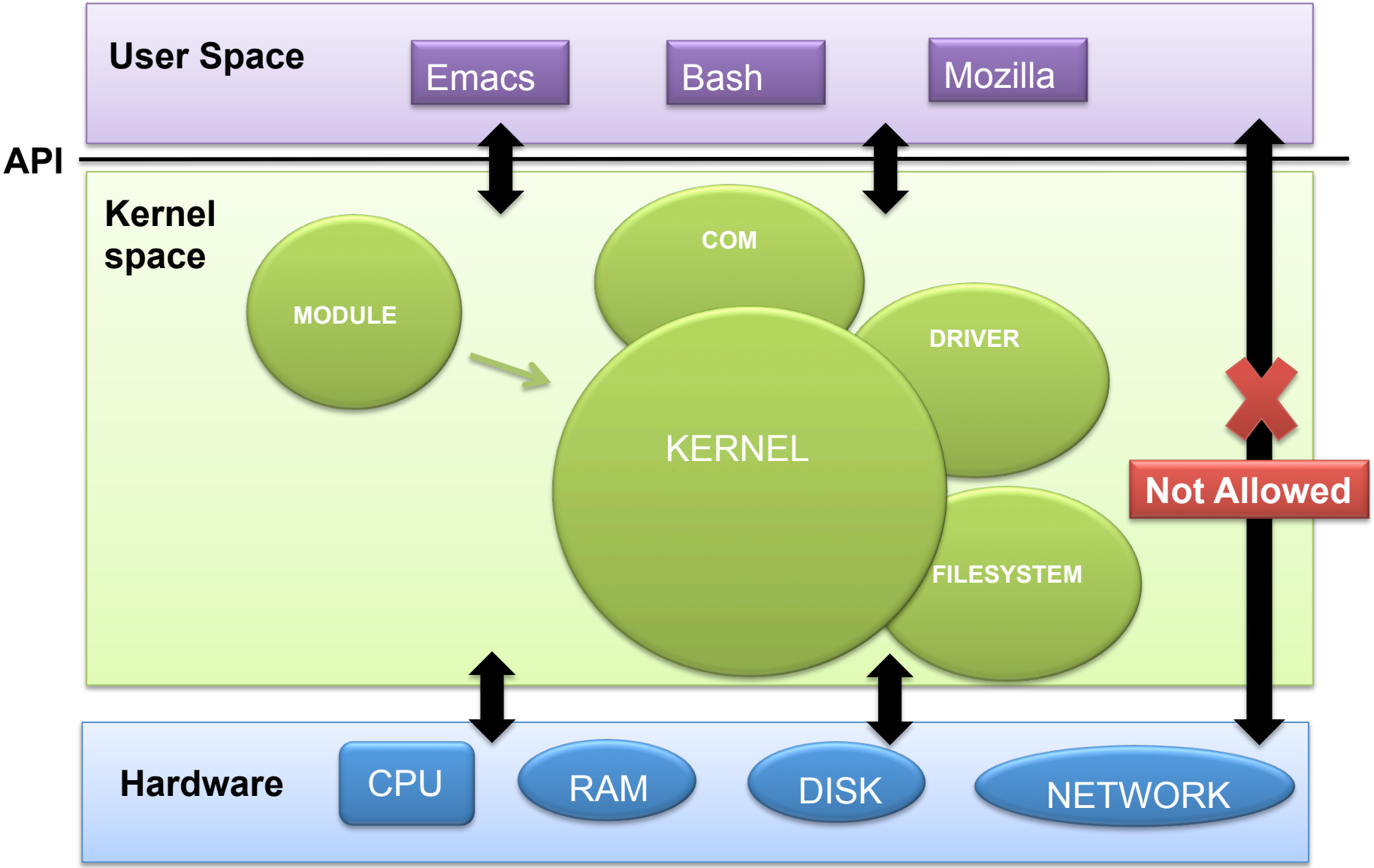
- Simulation
- Debug
- Execution

II. WHY NOT TO CHOOSE LINUX ?

LINUX

- Linux is an open source OS
- Manage I/O
  - Disk
  - IHM (keyboard, mouse, screen)
  - Network
- Multitask
  - Load balancing
  - Resource management
  - Average response time

Linux kernel and Modules



The Linux KERNEL

- Functions
  - scheduling functions and priorities
  - processus management
  - Timers ans alarms, time functions
  - Hardware resources access
  - Interrupt handling
  - User and group management
  - Modules management
  - Debug functions (printk)
  - Exceptions handling
  - System information

The kernel is not multitask and can’t be preempted

Linux Modules

- Why modules ? **Modularity !!!**
  - ⇨ Avoid kernel recompilation when drivers are added
  - ⇨ Save memory : only needed functions or drivers can be loaded
- A module can use functions, variables and kernel strucutres
- Code is executed with administrator privileges
- A module is allowed to acces I/O

Modules

- Modules are referenced in a list updated by the kernel (`lsmod` to vizualise this list)
- A module can only use functions exported by the kernel or other modules
- A module can only be loaded and removed by the user *root*
- The kernel updated a list of linked between modules (it allows to know when a module can be removed)

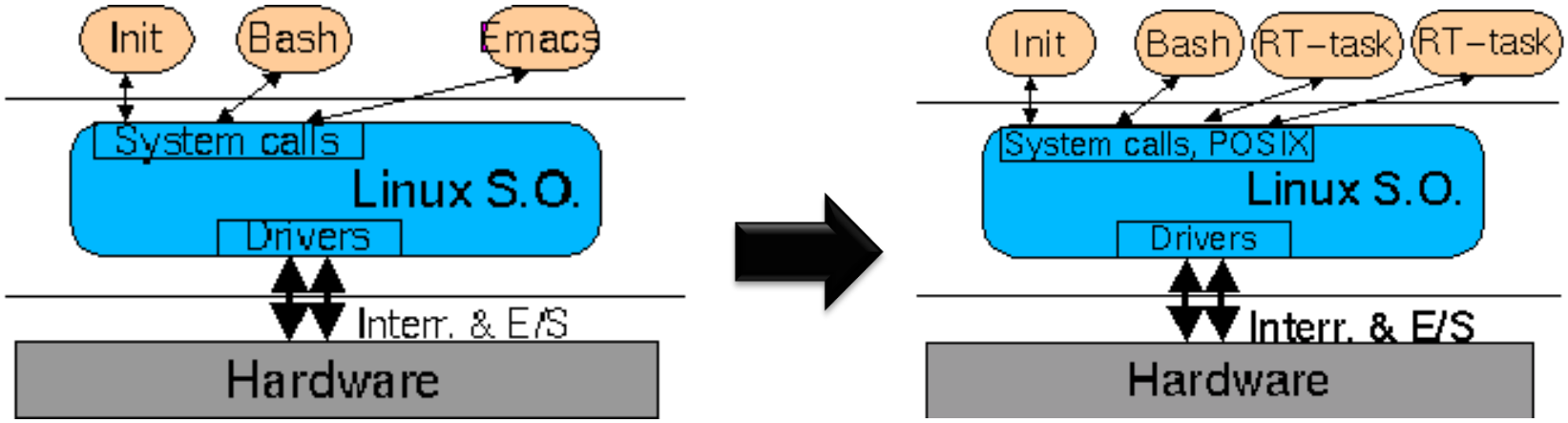
Linux and Realtime

- Process
  - wake up date can’t be guaranted
  - Round robin scheduler : load balancing
  - Calls to kernel functions can not be preempted
- Memory
  - Swap on Ram and process
- Interrupt
  - Regularly disabled by the kernel

⇨ LINUX IS NOT REAL TIME

III. REALTIME LINUX

KURT implementation

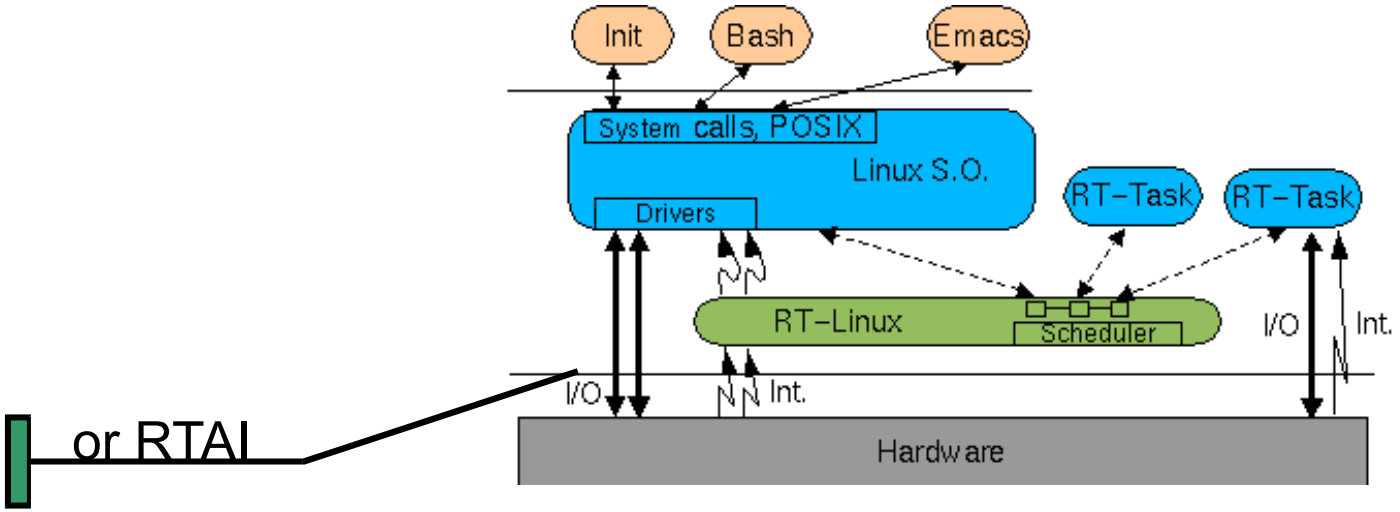


Linux **KURT**

- A realtime API is added in the kernel
- kernel preemption is allowed
- Interruption disable is reduced
- response time ≈ 100µs

! For soft realtime constraints applications

RT-Linux, RTAI



- A micro kernel is added under linux kernel
- Linux is executed in background
- Linux and a RTOS in a same system
- *https://www.rtai.org/*
- Response time < 20µs

Used in hard realtime constraints applications

Others realtime solutions around LINUX

- RTLinux
- RTAI (variante de RTLinux)
- eCos
- . . .
- And commercial distributions
  - FSMLAbs (RTLinux)
  - MontaVista HardHat
  - Lineo Embedix Realtime
  - TimeSys Linux/RT
  - LynuxWorks BLueCat Linux (Lynx Real Time Systems)

RTAI

- Real time small executive
- New functions can be added
  - Multi schedulers
  - Communications
- Open source
- Split in sevral modules
- Manage flotting point unit (FPU)

RTAI lives with LINUX

- Linux is unchanged : all linux applications can still be used

- Clear separation between realtime (RTAI) and non realtime (Linux) behaviour :
  - RT tasks : no linux system calls
  - RT tasks and linux application communications
    - Memory sharing
    - Real time Fifo

IV. USING RTAI

How to build a realtime application ?

- Modify the Linux kernel
  - RTAI installation, guidelines
    - Patch Linux
    - Compile the patched kernel
    - Install and run the new kernel
- Write the realtime application in module
  - Executed in kernel space
  - Incorporates realtime tasks
  - 1 or more module
- Execute the application
  - Compile the application modules
  - Load all the needed RTAI modules
  - Load the application modules

Write a module (linux 2.4)

```
#define MODULE
#include<linux/init.h>
#include <linux/module.h>
#include<linux/version.h>

static int output=1;

int init_module(void) {
   printk("Output= %d\n",output);
   return 0;
}

void mon_code(void) {
  output++;
}

void cleanup_module(void){
   printk("Adiós, Bye, Ciao, Ovuar, \n");
}
```

Write a module (linux 2.6)

```
#define MODULE
#include<linux/init.h>
#include <linux/module.h>
#include<linux/version.h>     no more needed

static int output=1;

int mon_init(void) {
   printk("Output= %d\n",output);
   return 0;
}

void mon_code(void) {
  output++;
}

void mon_cleanup(void){
   printk("Adiós, Bye, Ciao, Ovuar, \n");
}
module_init(mon_init);
module_exit(mon_cleanup);
```

Compiling and executing

- Compile a module :
  - Linux 2.4 : `gcc -I /usr/src/linux/include/linux -O2 -Wall -D__KERNEL__ -c exemple1.c`
  - Linux 2.6 : kernel source and makefile needed
- Load a module

```
> insmod exemple1.ko (exemple1.o with kernel 2.4)
```

- Printk output on console

```
> dmesg | tail -1
Output= 1
```

- List modules

```
# lsmod
Module      Pages      Used by:
exemple1        1           0
sb              6           1
```

Load steps (resumed)

1) External referenced symbol verification
2) Kernel version verification
3) Code and data memory allocation and copy of the module in the allocated space
4) Symbol table update with module symbols (`output` and `mon_code`)
5) Call of module `init_module` function

Cleaning

- Remove module
  - Will execute the `cleanup_module()` function

```
# rmmod exemple1
# dmesg | tail -2
Output= 1
Adiós, Bye, Ciao, Orvua,
```

Pass data to a module ?

```
# insmod exemple1.ko output=4
# dmesg | tail -3
Output= 1
Adiós, Bye, Ciao, Orvua,
Output= 4
```

Makefile

- Interpreted by the command `make`
- Set of **rules**
- Rule : define commands to build a **target** from source files. Also define **dependancies** (source files) needed by the commands

```
rule : dépendancies
       command
```

- www.gnu.org/software/make/manual

RTAI modules

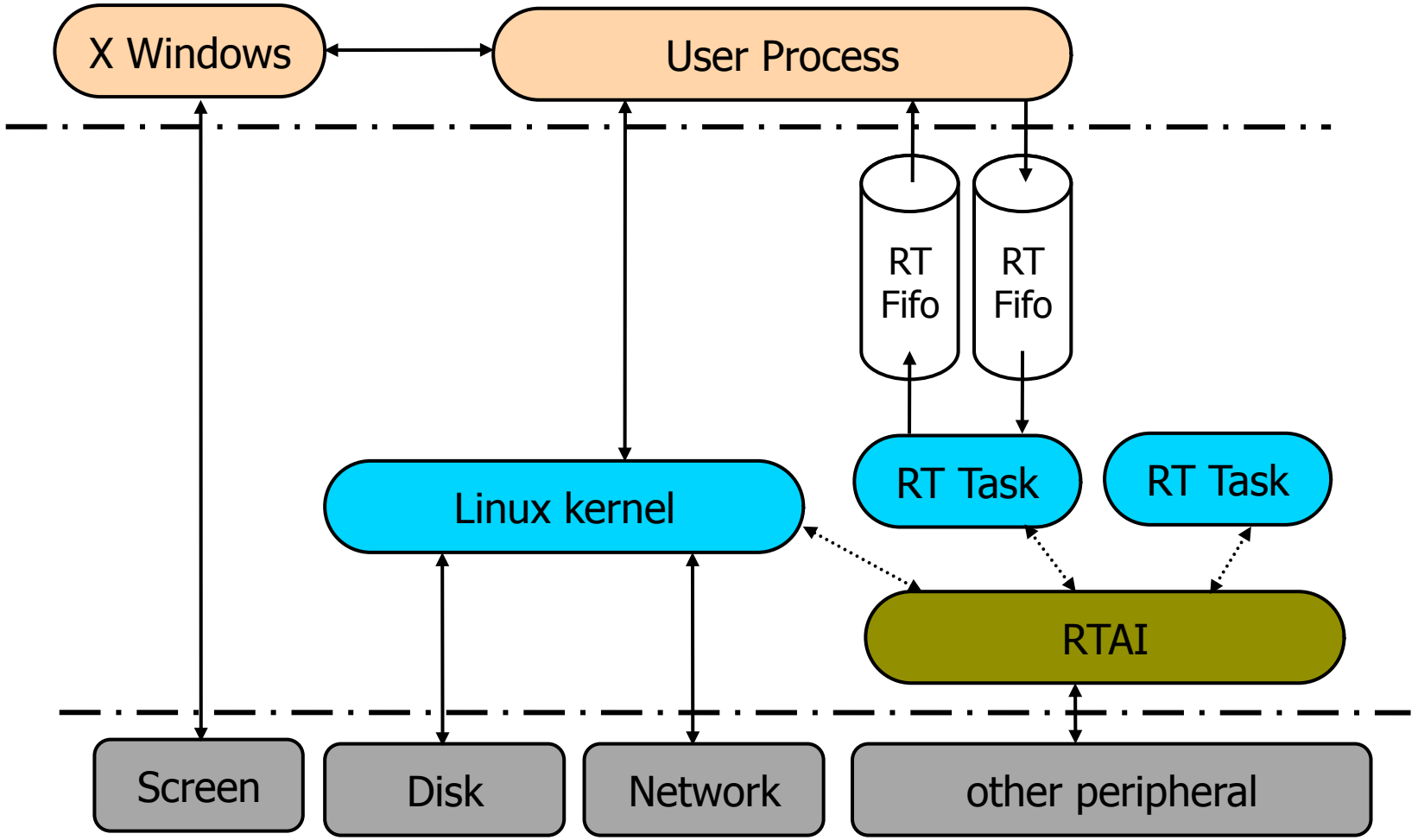
- `rtai_hal` : base functions
- `rtai_sched` : mono processor scheduler
- `rtai_sem` : semaphore functions
- `rtai_fifos` : RT modules and linux applications communications functions

+ application modules
  - I/O handling
  - Tasks

- To be loaded each time the application has to be started

⇨ Script

Typical Realtime application



V. RT TASK SCHEDULING WITH RTAI

RTAI : task handling

- RTAI
  - Each Realtime task is a thread `RT_TASK`
  - Linux kernel = idle priority thread

- Task scheduling
  - Preemptive
  - Fixed priority (default) or EDF
  - ! Processor overload => Linux can not be executed

Task create

```
#include <rtai_sched.h>
MODULE_LICENSE("GPL");
static RT_TASK tache;
#define STACK_SIZE 2000

iret = rt_task_init(&tache,  // task pointer
                    routine, // name of function to be execute
                    1,       // int  passed to routine
                STACK_SIZE,  // execution stack size
                    1,       // int priority, 0=highest priority
                    0,       // int 0 => fpu not used
                             //       1 => fpu is used
                    0        // signal (see rt_task_signal_handler)
                      );
```

- iret = 0 if ok, <0 either
- task creation is done in `init_module`

Periodic task

- In `init_module`

```
iret = rt_task_make_periodic(&tache,
           start_date, nano2count(PERIODE));
```

⇨ Periodic execution,first execution since `start_date`, units is `count`

- In the task

loop containing task **`+ function`** `rt_task_wait_period ();`

⇨ suspend task execution until next period

Suspend and delete a task

- Suspend

```
rt_task_suspend( rt_whoami() );
```

- Re-execute

```
rt_task_resume( &tache);
```

⇨ Also allows to execute a non-periodic task

- Ending

```
rt_task_delete( &tache);
```

Time

- Timer : 2 possible behaviour
  - `rt_set_oneshot_mode()`
    - Recent Pentium provides CPU TSC (Time Stamp Clock) => variable time base
    - 386,486 and old pentium don’t provide CPU TSC => need to read an external timer 8254 => cost time, not efficient
  - `rt_set_periodic_mode()`
    - Default mode
    - Fixed period
    - The tasks whose period is not a multiple of timer period are executed with the nearest multiple period

Time (2)

- First, the timer must be started

```
tick_per = start_rt_timer(per);
```

  - one shot mode : `per` irrelevant
  - Periodic mode : `per` = period of the timer in count,

    `tick_per` = alocated period

- To read the timer

```
rt_get_time();   in count unit
rt_get_time_ns();   in nanosecond unit
```

- Conversions

```
RTIME tn, tc;
tc = nano2count(tn);
tn = count2nano (tc);
```

- Stop the timer (cleanup_module)

```
stop_rt_timer();
```

RTAI scheduling

- Scheduling policy can be combined
  - EDF > fixed priority
  - Linux = task with the smallest priority
- Priority
  - Fixed priority : de 0 à `RT_LOWEST_PRIORITY`

```
ma_prio = rt_get_prio( rt_whoami());
old = rt_change_prio(rt_whoami(),
                     ma_prio+1);
```

  - Dynamic priority

```
rt_task_set_resume_end_times(wakeup_date, duedate_date);
rt_task_set_resume_end_times(-period,-deadline);
```

Multiprocessor scheduling

- SMP : Symetric Multi Processor
- 2 solutions
  - 1 scheduler handle n processors
    - Execute the n highest priority tasks among the t ready tasks
  - 1 scheduler on each processor
    - Each task is allocated to a processor

VI. SYNCHRONISATION, COMMUNICATION

Synchronisation / Communication: why?

- Transmit an event
  - fugitive, memorised, incremented
  - With/without value
- Ordering
  - before / after
  - Control the number of execution
- Data exchange
  - With synchronisation
  - Without synchronisation
    - Ensure data integrity

RTAI : binary semaphore - mutex

- Rendez-vous, mutual exclusion
- Creation

```
SEM mon_sem;
rt_typed_sem_init(&mon_sem,  // semaphore pointer
                          1, // initial value
                  BIN_SEM    // type
                          );
```

- Take a semaphore

  If the semaphore is présent : on prend le sémaphore et on continu

  Si le sémaphore est absent : on est bloqué jusqu'à ce que le sémaphore soit restitué

```
ierr = rt_sem_wait(&mon_sem);
```

- Restituer le sémaphore

```
ierr = rt_sem_signal(&mon_sem);
```

Sémaphores (2)

- Problèmes : inversion de priorité, interblocage
- Autres types

```
rt_typed_sem_init(&mon_sem,1,BIN_SEM);
```

  - A compte : CNT_SEM
  - Ressource : RES_SEM
    - met en place le protocole de priorité plafond

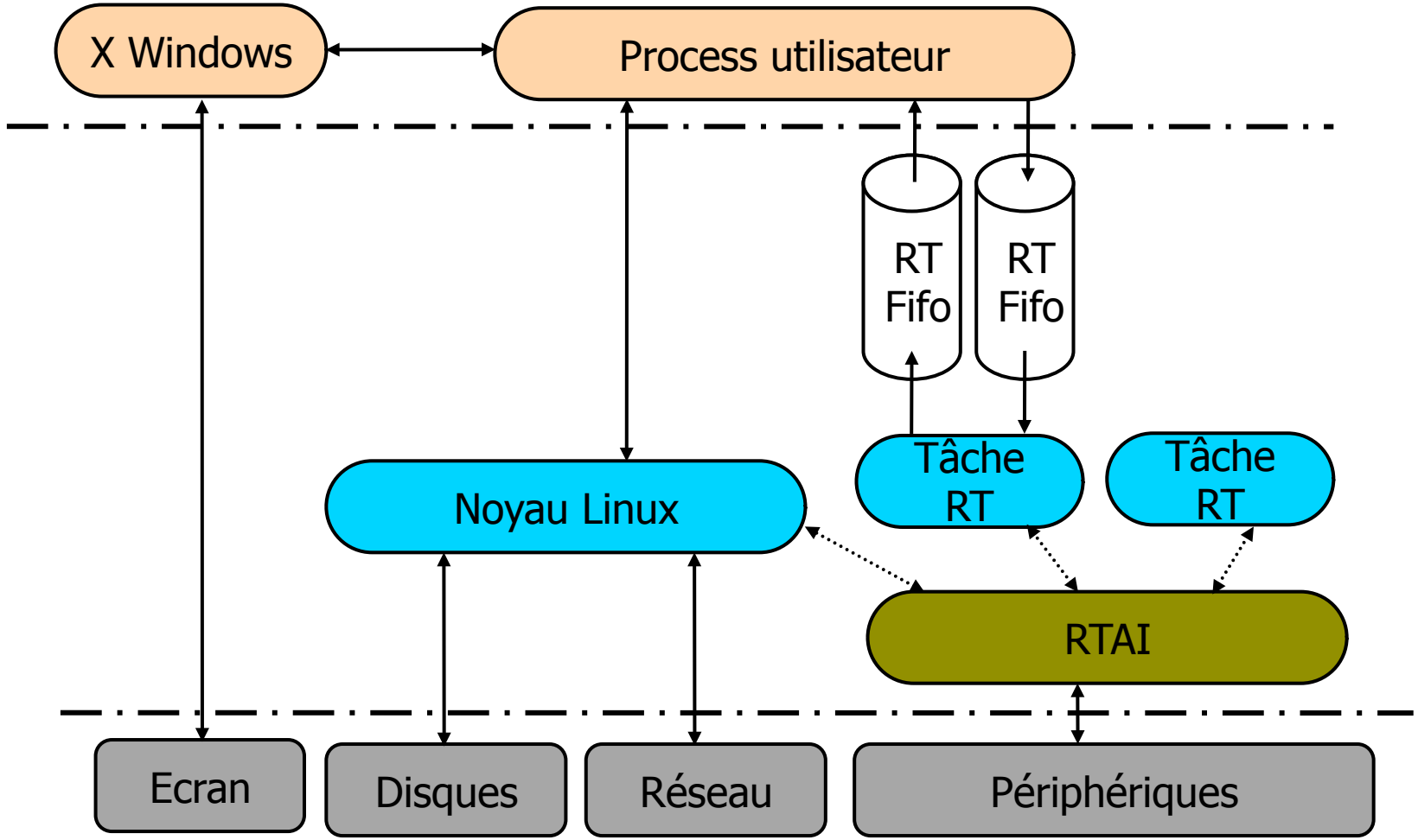
- Destruction

```
- ierr = rt_sem_delete(&mon_sem);
```

- Variantes asynchrones

- `rt_sem_wait_if` : pas de blocage si sémaphore absent
- `rt_sem_wait_until(&sem,date)` : blocage au maximum jusqu'à `date`
- `rt_sem_wait_timed(&sem,durée)` : blocage au maximum pendant `durée`

RTAI : Fifos


X Windows
Process utilisateur
RT Fifo
RT Fifo
Noyau Linux
Tâche RT
Tâche RT
RTAI
Ecran
Disques
Réseau
Périphériques

RTAI : RTFifos coté process utilisateur

- côté process utilisateur
  - périphérique spéciaux de type caractère :
    - /dev/rtf0, /dev/rtf1 v…
    - accessibles à travers les appels : open, ioctl, read, write
  - communication sans structure
    - file FIFO tampon d ’un nombre d ’octets fixé
  - n ’existe (ne peut être ouvert) que si déjà créé par tâche RT ou par appel spécifique

RTAI : RTFifos coté RT

- 2 modes d 'utilisation
  - comme périphérique
  - API spécifique RT

`rtf_create(unsigned int` ***fifo***`, int taille);`

création,

périphérique `/dev/rtf`***`fifo`*** existe,

`rtf_destroy(unsigned int` ***`fifo);`***

destruction,

mémoire réallouée

`rtf_put(fifo, char *tampon, int compte);`

`rtf_get(fifo, char *tampon, int compte);`

retour -1, si insuffisamment de données ou

de place

non bloquant

RTAI : RTFifos

- côté tâche RT (2)
  - API spécifique RT

```
int my_handler(unsigned int fifo);
```

```
rtf_create_handler(3,&my_handler);
```

routine `my_handler` exécutée en mode noyau quand des données sont lues ou écrites

  - permet de réveiller une tâche RT, via p.ex. un task_resume
  - permet d ’éviter la scrutation régulière de la FIFO
  - permet de mettre en place un schéma ASYN/SYN quelconque avec sémaphores

`rtf_resize =>` redimensionner la fifo

- plus d ’autres fonctions

Messages

- Caractéristiques
  - Communication point à point (tâche à tâche)
  - message : valeure entière
  - SYN/SYN
  - `ierr = rt_send(&tache_but,valeur); //SYN`
  - ```
    ierr = rt_receive
           (&tache_orig,&entier); //SYN
    ```
- Variantes
  - `If :` send ASYN, pas d'envoi si recepteur non prêt, pas de blocage de l'emetteur
  - `timed et until :` blocage limité dans le temps
  - `rt_receive(0 … :` n 'importe quel émetteur
- Pas d 'initialisation spécifique
    ```

Boîtes aux lettres (Bal)

- Caractéristiques
  - Communication multi/multi point
  - message : taille variable
  - ASYN/SYN

  - `ierr = rt_mbx_send(&Bal,&msg,taille_msg); ASYN,` mais blocage tant que le message entier n'a pas été copié dans la Bal
  - `ierr = rt_mbx_receive(&Bal,&msg,taille_msg);` Blocage tant que le message n'a pas été reçu

- Variantes send
  - `if :` envoi ssi le message entier peut être copié dans la bal
  - `Wp :` envoi autant d'octets que possible sans blocage
  - `timed et until : blocage limité dans le temps`
- initialisation spécifique
  - `ierr = rt_mbx_init(&Bal,taille)`
  - `ierr = rt_mbx_delete(&Bal);`

Le problème de l ’allocation mémoire

- Segmentation,
- Mémoire paginée
- pas new/delete ou malloc/free
  - kmalloc / kfree

- **non temps réel**

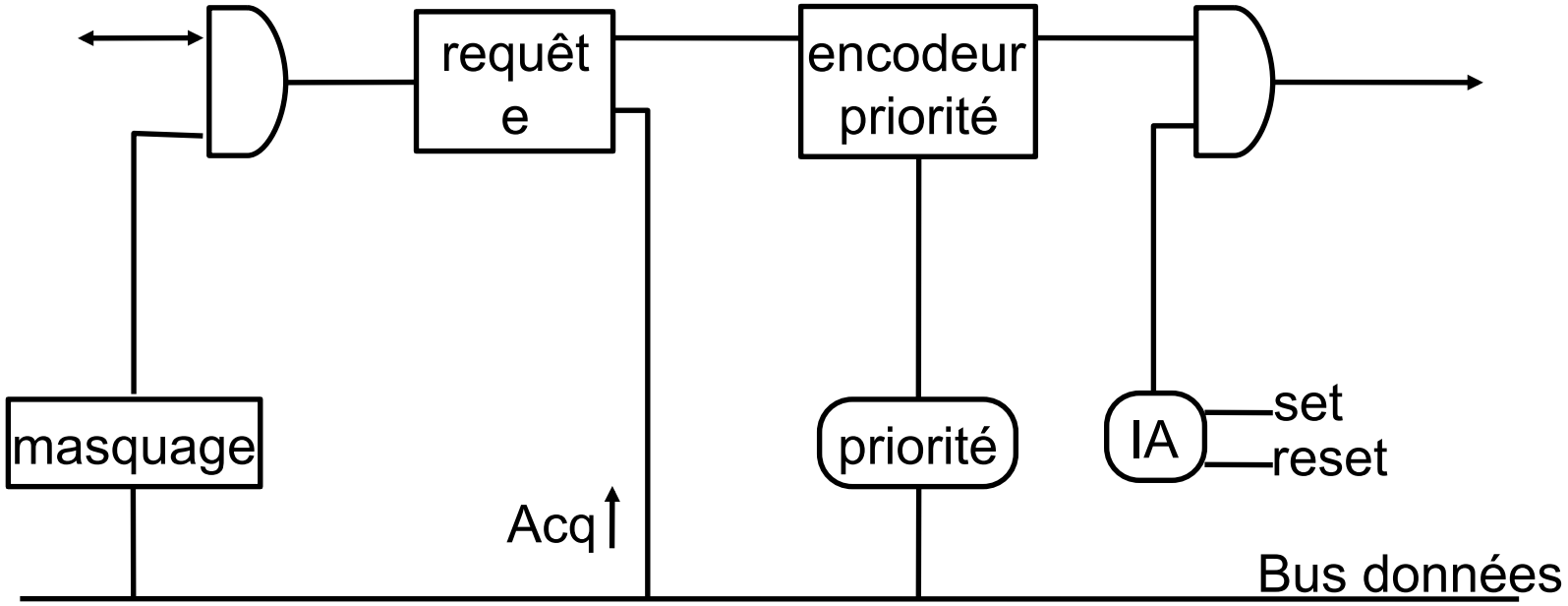
- **solutions RT :**
  - s ’allouer la mémoire de manière statique
  - allocation dans une table de blocs limitée

- utilisation de kmalloc au niveau noyau

RTAI : Interruptions

- permet réactions rapides
- inhibe le noyau temps réel
- niveau matériel et niveau logiciel
- gestionnaire d ’IT simples avec temps de réponse fixe (ou borné)
  - prise en compte d ’un événement
  - phase urgente, laisser une tâche dans l ’espace utilisateur faire le travail de fond
- ITs/événements peuvent être émis par :
  - équipement informatique (clavier, écran, disque …)
  - équipement industriel (capteur, actionneur, alarme …)
  - horloge : donne le rythme (tick = période d échantillonnage du système)

Interaction par Interruption



- rt_global_cli() rt_global_sti()
- rt_disable_irq(), rt_enable_irq()

RTAI : Interruptions (2)

- Installation

```
#define IT_CLAVIER 1
 rt_request_global_irq(IT_CLAVIER,
 mon_gestionnaire);

  rt_free_global_irq(IT_CLAVIER);

void mon_gestionnaire(void) {
 rt_pend_linux_irq(IT_CLAVIER);
```

- Utilisation
  - traitement urgent
  - réveiller tâche endormie

RTAI : communication avec le matériel

- par pilote de périphérique (`posixio`)
  - « fichiers » `/dev/nom`
    - open/ close/ read/ write
- accès direct aux ports E/S

```
#include <asm/io.h>

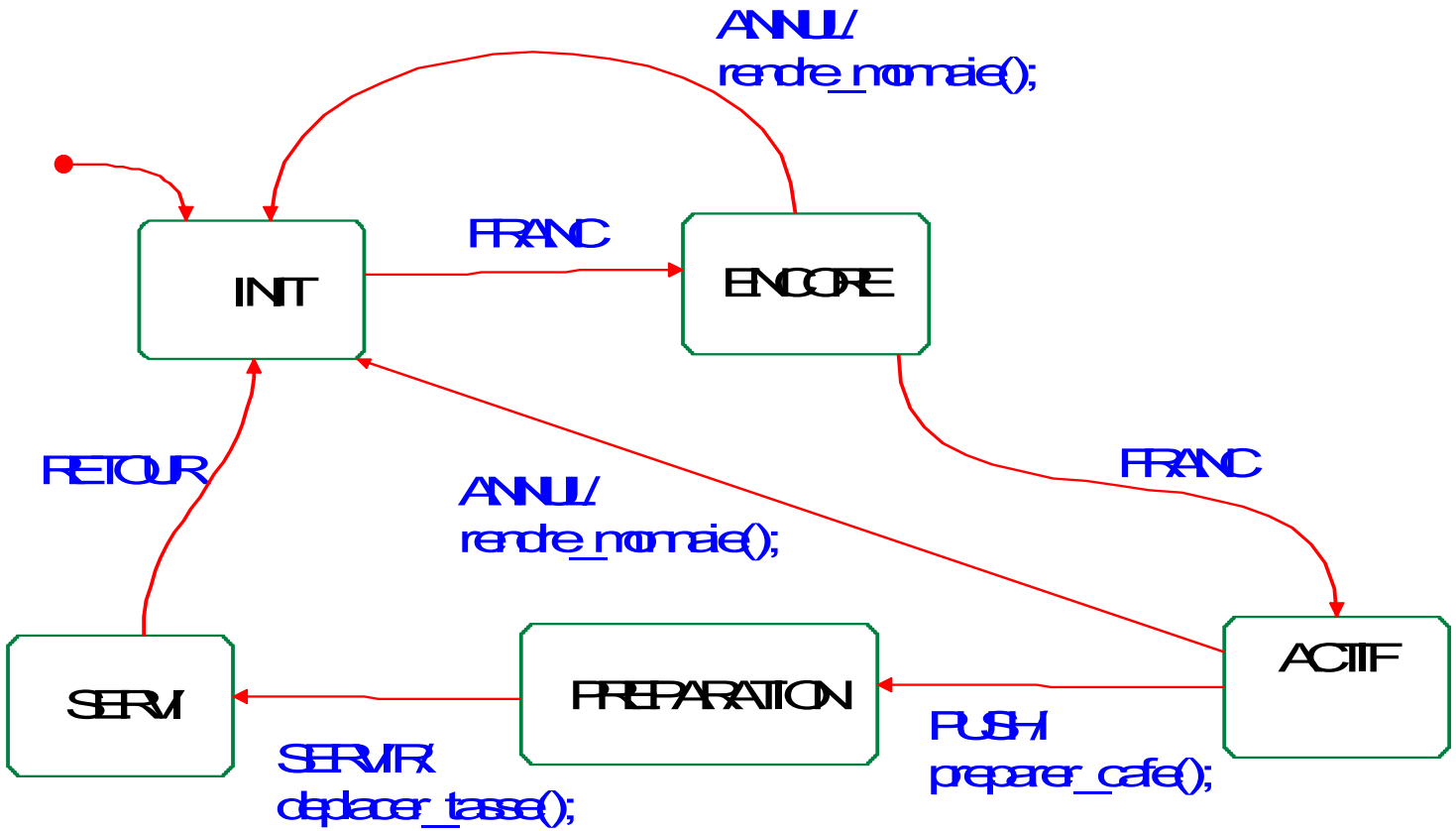

outb(valeur, adresse)
valeur = inb(adresse)
```

    - taille du mot : b, w, l
    - pause : _p : outl_p

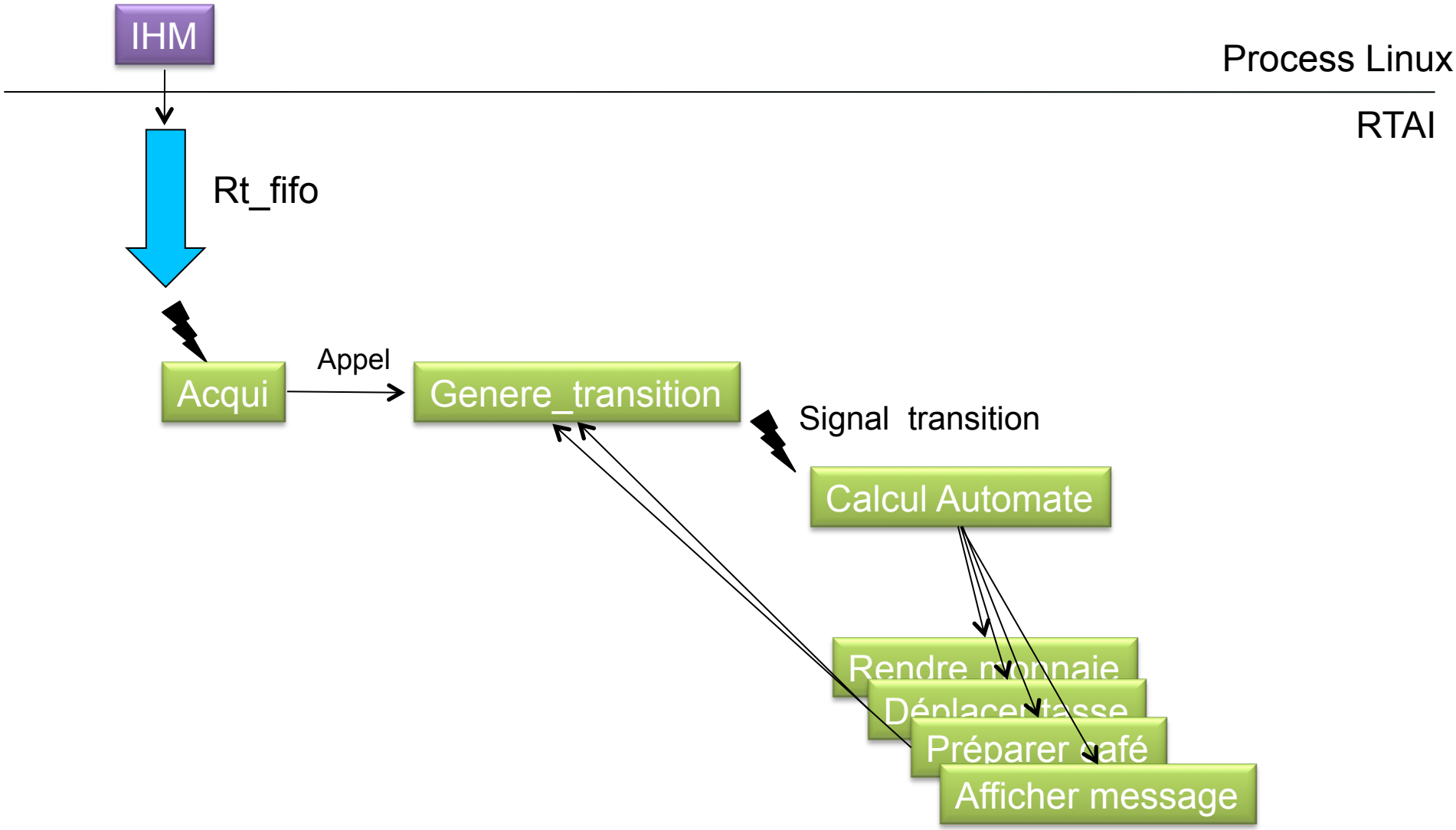
Schémas classiques

- Chien de garde (Watchdog)
- Comment gérer un automate d ’état (machine à café, menu téléphone …)
- BD : lecteur rédacteur
- Client Serveur
- tableau noir
- …

Exemple : machine à café



Machine à café : spcéfication fonctionelle



VII.RTAI : CONCLUSION

RTAI : quelques remarques

- évolution permanente
- possibilités en phase de mise au point non détaillées ici
  - outils de trace : LTT
  - exécution en mode user : LXRT
- Xenomaï
  - POSIX, VxWorks,VRTX,pSOS+,µITRON,RTAI

RTAI : quelques remarques

- évolution permanente
- possibilités en phase de mise au point non détaillées ici
  - debug
  - outils de trace : LTT
  - exécution en mode user : LXRT
- Xenomai

RTAI-LAB

Spécification schémas-blocs

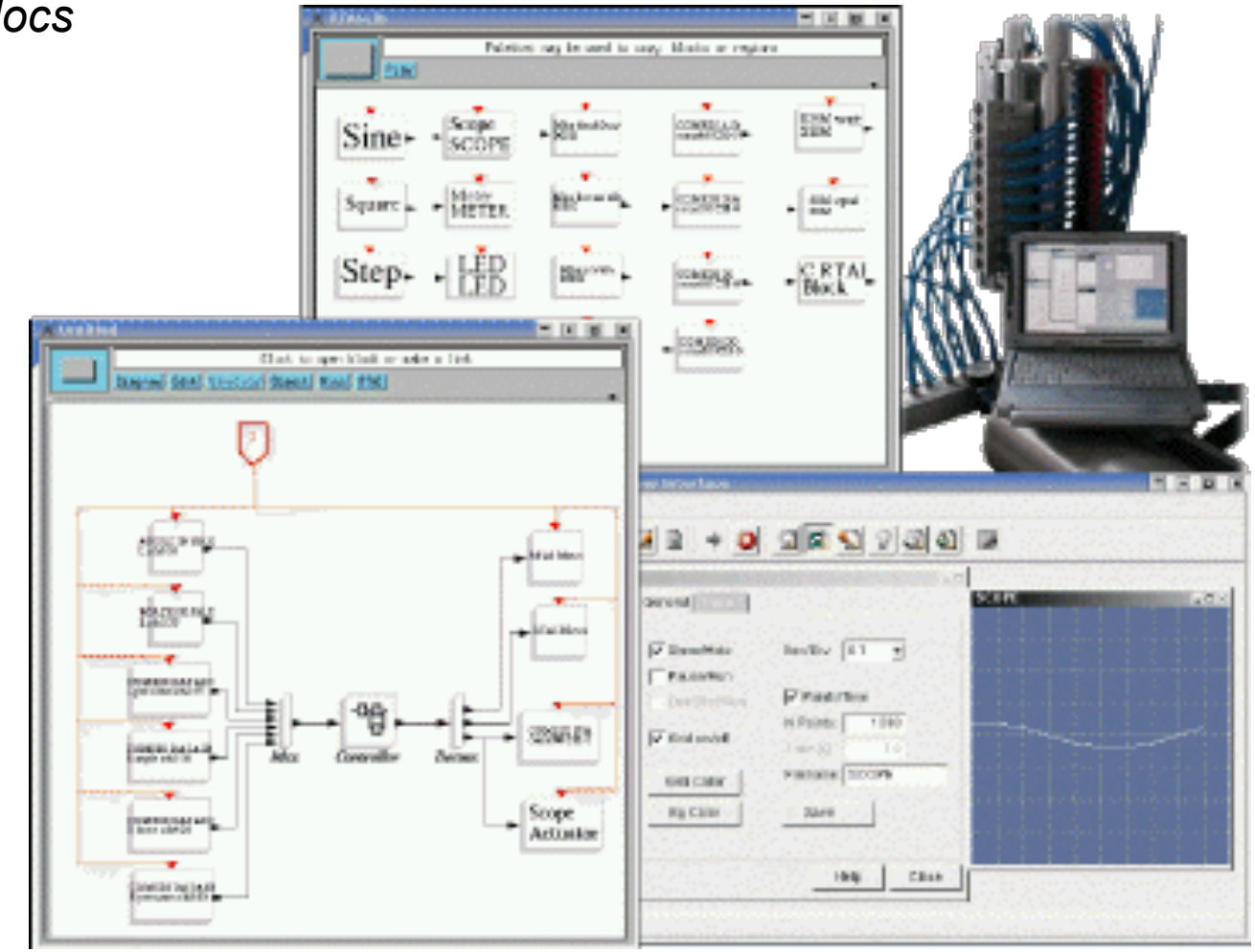
SCILAB/SCICOS

MATLAB/SIMULINK

↓ Génération code + Librairie RTAI-LAB

RTAI

Exécution



http://www.rtai.org

http://www.scicos.org/

http://www.mathworks.com/products/simulink/

VIII. AND OTHER RTOS …

Some RTOS … (http://www.dspconsulting.com/rtos.html)

RTOS	URL	Processor	Licensing
Tornado, VxWorks	http://www.wrs.com	x86, 68k, PPC, CPU 32, i960, SPARC, SPARCLite, SH, ColdFire, R3000, R4000, C16X, ARM, MIPS	$16,500 per user + target
Hard Hat Linux	http://www.mvista.com	x86, PPC, MIPS and ARM	No
VRTX	http://www.mentor.com	x86, 68k, PPC, i960, ARM	$2,000 par user + target
OS-9 V3.0	http://www.microware.com	x86, 68k, PPC, SH3, StrongARM	? + target
QNX	http://www.qnx.com	AMD ÉlanSC300/ 310/ 400/ 410, AM386 DE/SE Intel386 EX, Intel486, ULP Intel486, Pentium, Pentium Pro, and NatSemi NS486SXF.	Pricing is based on OS modules used on the target system and per unit shipped.
pSOS	http://www.isi.com	86, 68k, PPC, 683xx, CPU32(+), MIPS R3000/4000/5000, Coldfire 510x/520x, i960, ARM 7 (TDMI), SH1/2/3, M32R	$17,000 per user + target
RTX	http://www.vci.com	x86, P5, P6	$150/RTX licence + target
LynxOS	http://www.lynx.com	x86, 68k, PPC, microSPARC, microSPARC II, PA-RISC	US$10,000 per user Licence: US$7000 + target

…. + many other

RTOS market survey(2005)

(source : http://www.embedded.com/)

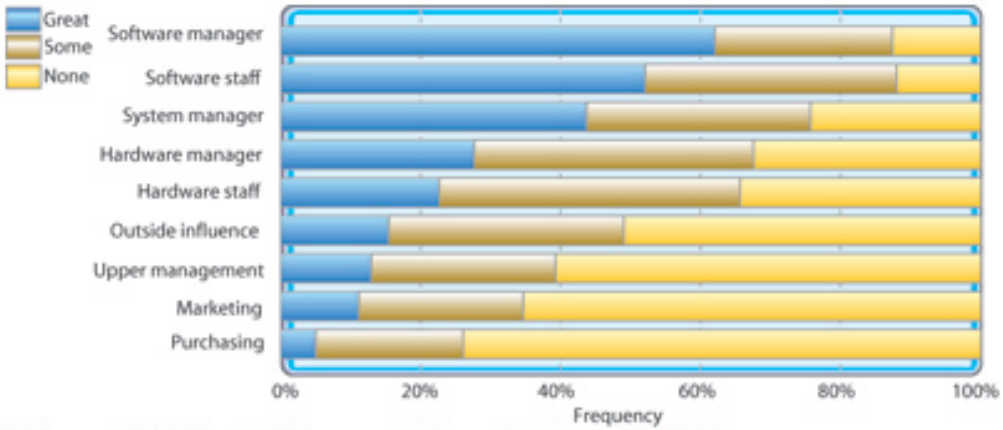


Figure 1: Who influenced the choice of OS?

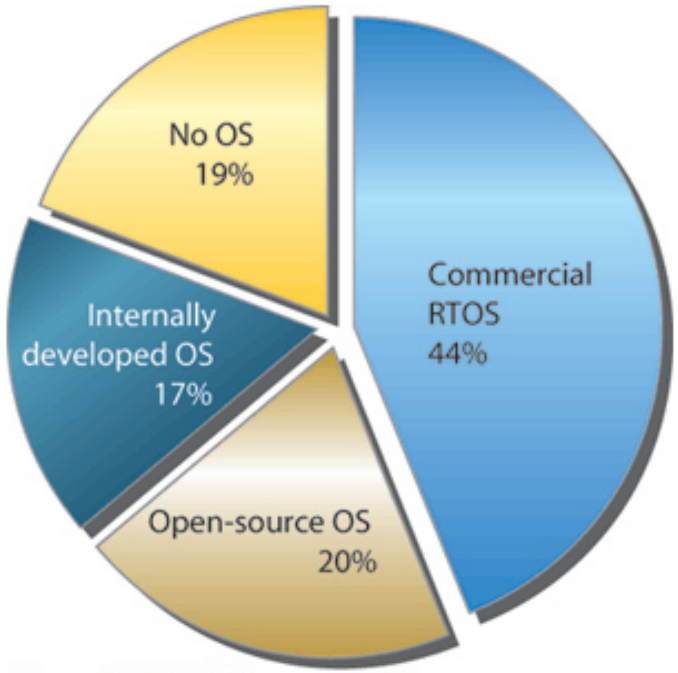


Figure 2: What type of OS?

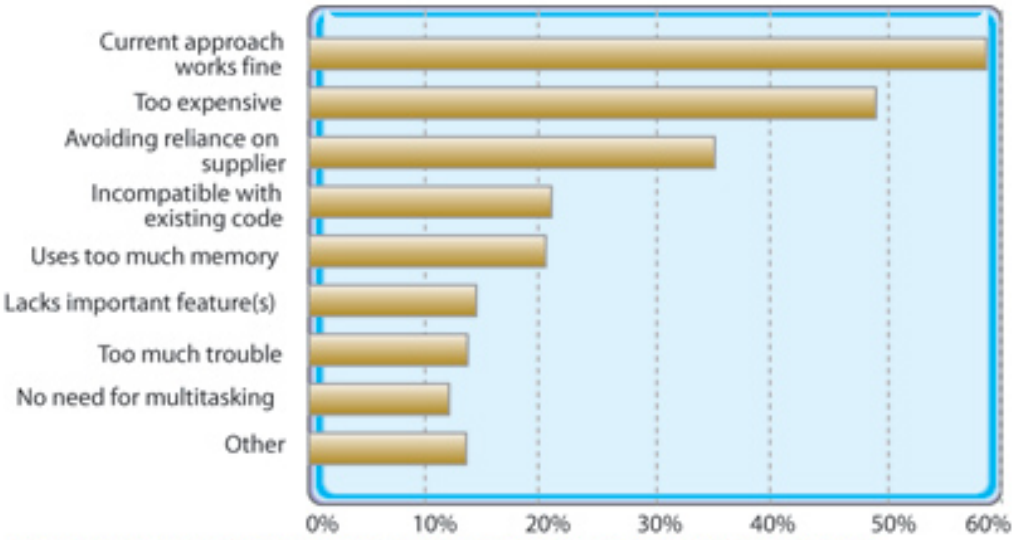


Figure 3: Reasons for not choosing a commercial OS

RTOS market survey

(source : http://www.embedded.com/)

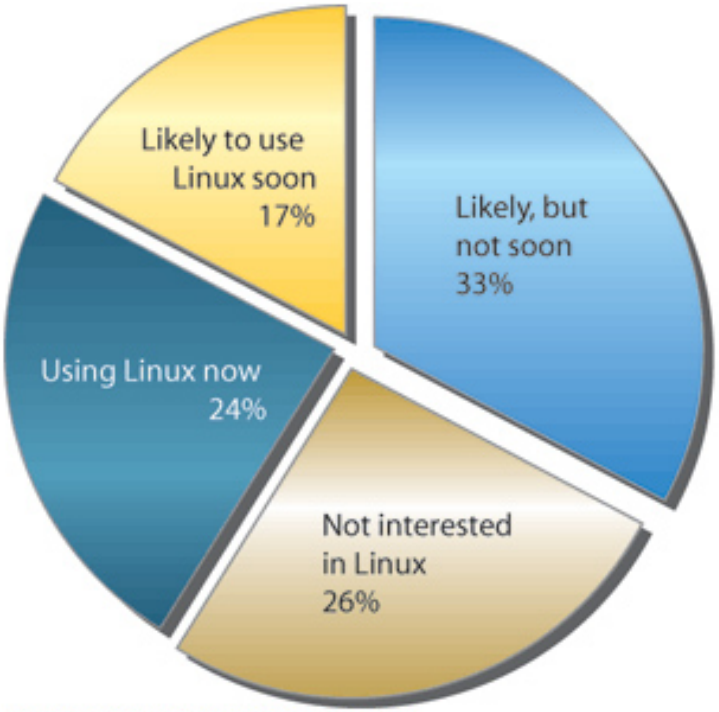


Figure 6: Interest in Linux

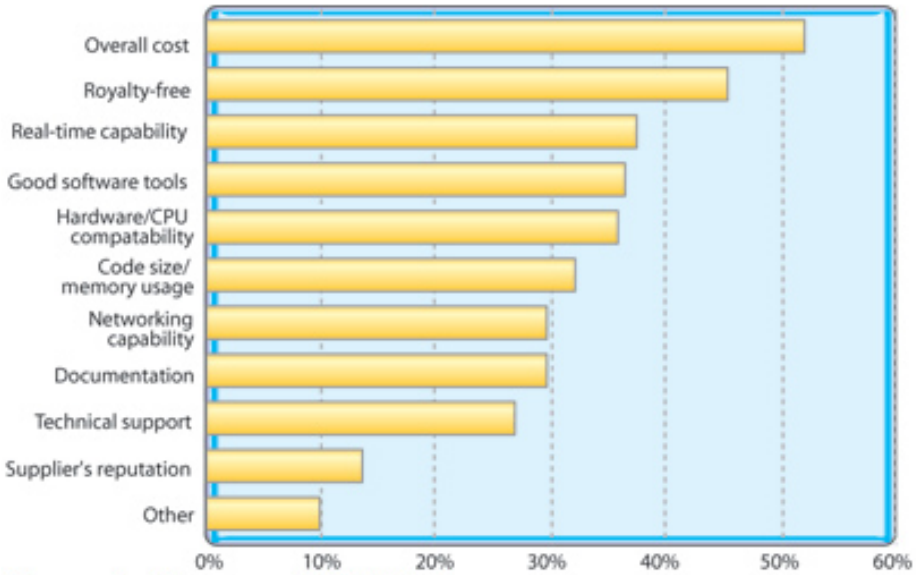


Figure 4: Commercial OS factors

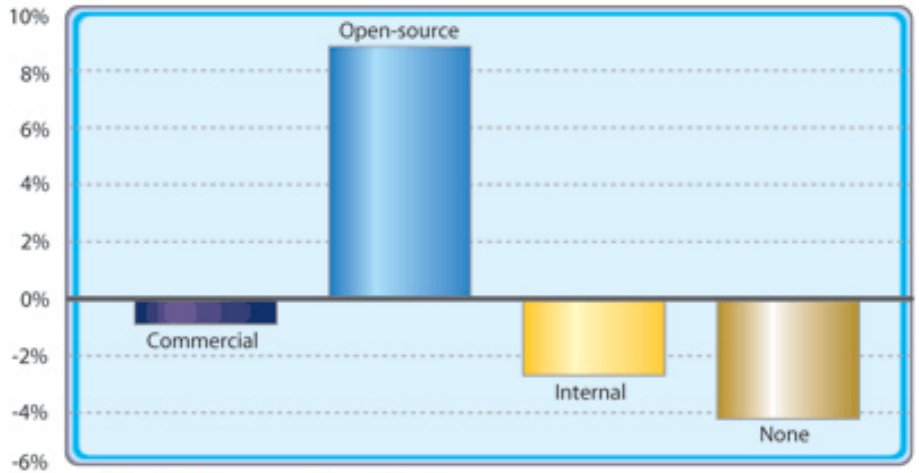


Figure 5: OS for next project

RTOS market survey

(source : http://www.embedded.com/)

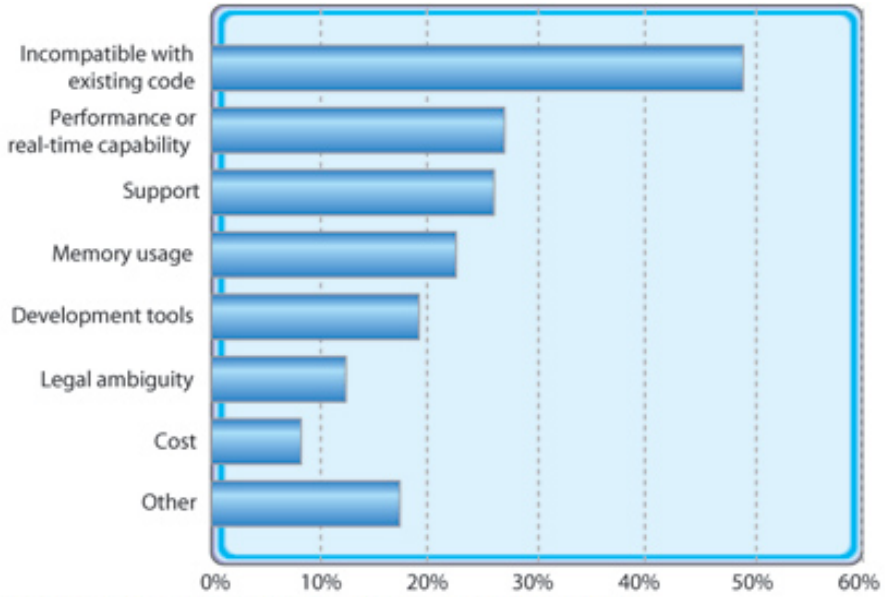


Figure 8: Reasons for not considering Linux

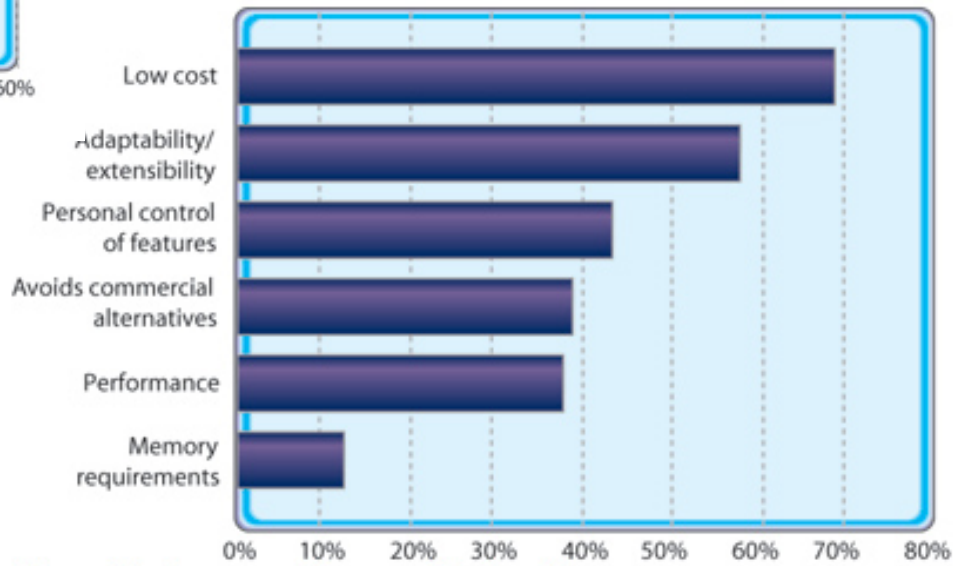


Figure 7: Reasons for considering Linux

RTOS market survey

(source : http://www.embedded.com/)

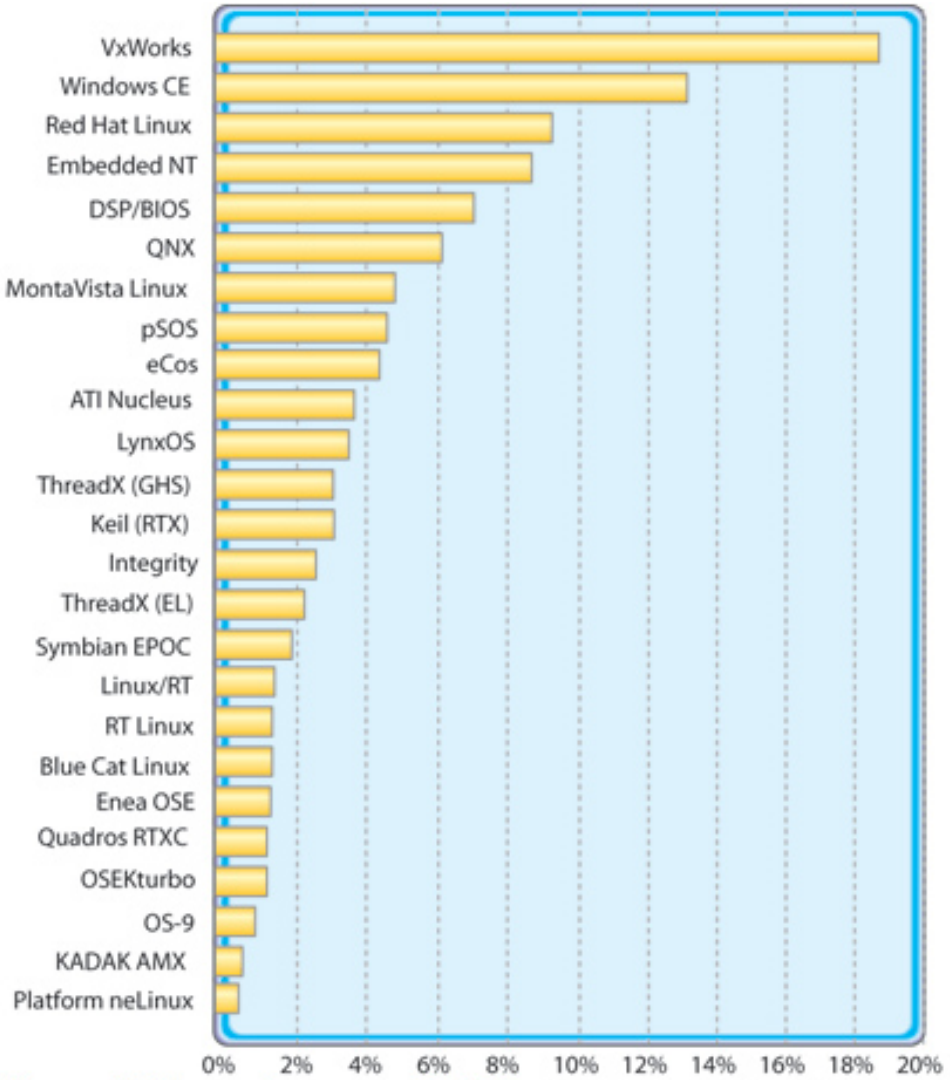


Figure 9: Current commercial OS

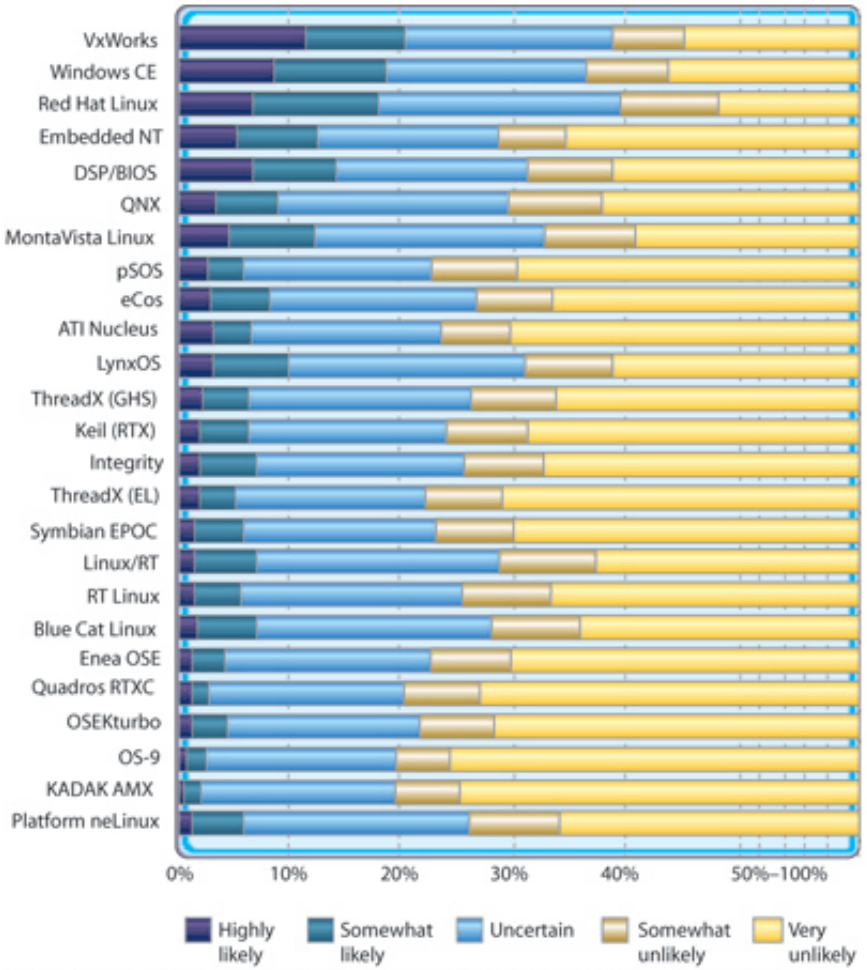


Figure 10: Commercial OS respondents would consider

Choice of an RTOS

- Performance = time to provide a service to the application, it depends on:
  - Hardware architecture,
  - What measuring ? => benchmarks
  - Préemption duration, Task switching, semaphore, memory allocation, services …
- Price : licence + per unit shipped
- Source Code, certification
- Tools