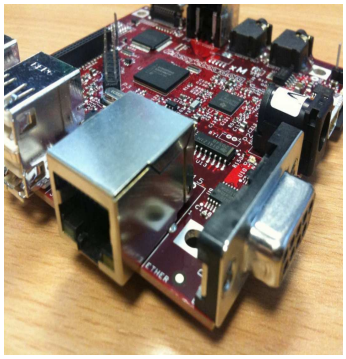
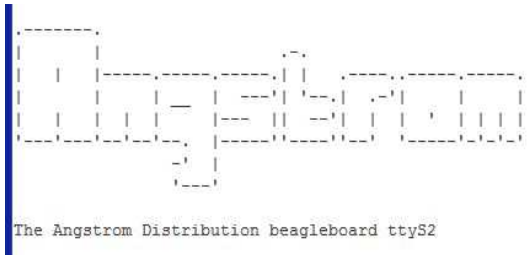


# Introduction

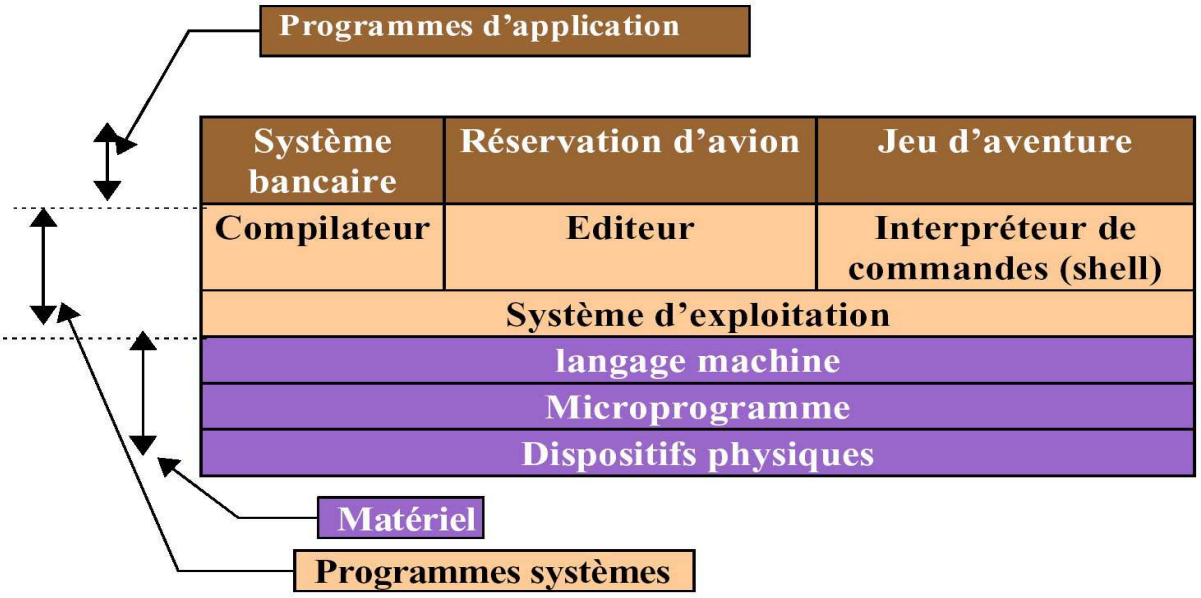
# A LINUX embarqué



# Introduction à LINUX embarqué

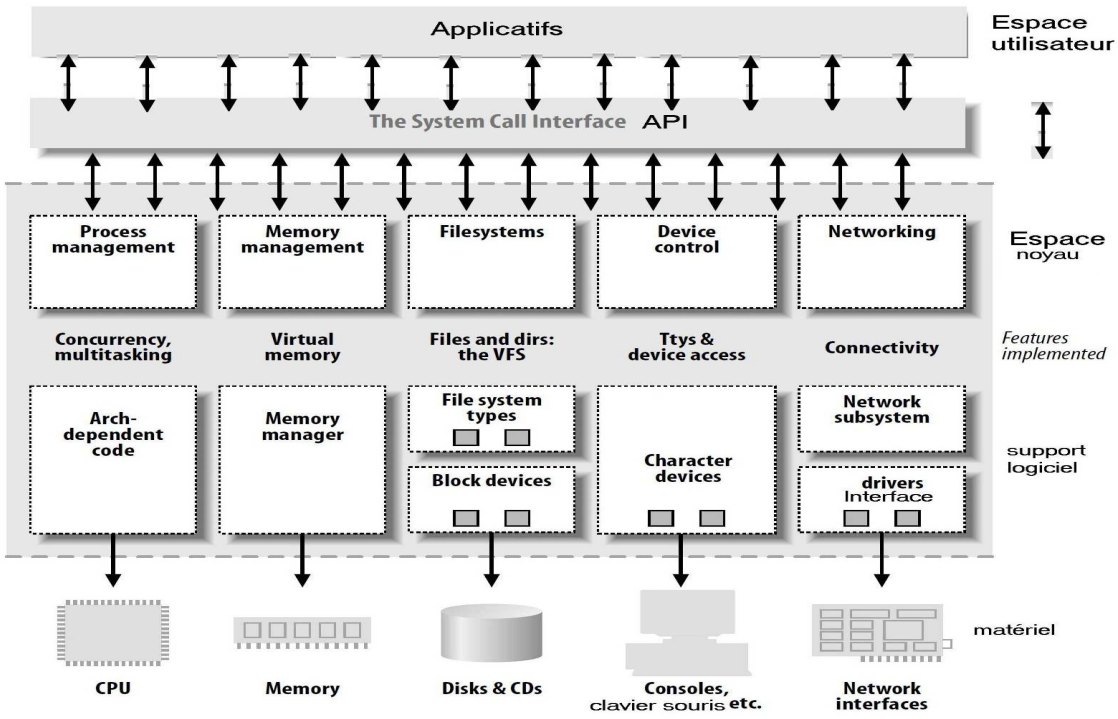
## 1) Le système d'exploitation

La complexité évidente du matériel implique la réalisation d'une **machine virtuelle** qui gère le matériel : **c'est le système d'exploitation**.




Le rôle principal du système d'exploitation est d'isoler les programmes des détails du matériel. Ils présentent à l'utilisateur une **« machine virtuelle »** ***plus facile à comprendre et à utiliser*** ; qui donne une image du matériel pas toujours fidèle à la réalité.

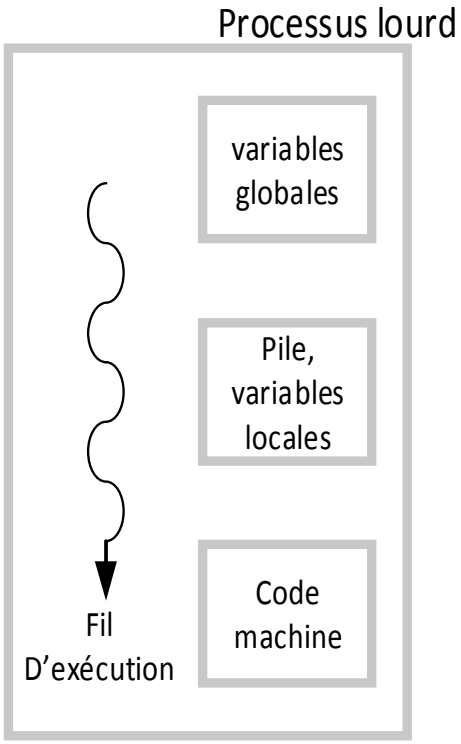
Pour rendre à l'utilisateur un accès plus simple à la machine le noyau du système d'exploitation a plusieurs fonctions :



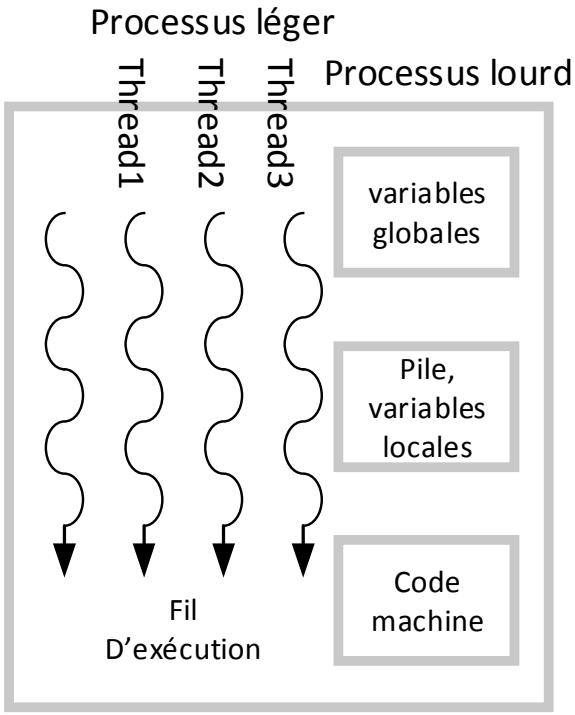
Le noyau va gérer :

- La mémoire MMU ou non
- Les systèmes de fichier
- Le réseau
- Les pilotes matériels
- Les processus lourds ou légers

**Un processus lourd** est un programme en cours d'exécution. Chaque processus lourd implique la gestion d'un espace d'adressage virtuel MMU et de nouvelles copies de toutes les variables et ressources nécessaires à l'exécution (pile, registres, fichiers ouverts, verrous etc…).
Le principal avantage de ces processus est la protection mémoire entre les processus du même système.
Lorsqu'on veut donner l'illusion d'une exécution en parallèle de plusieurs applications, on doit gérer plusieurs flots d'instructions, sachant que, à chaque instant, seulement un de ces flots sera exécuté sur chaque processeur. Traditionnellement, on appelle de tels flots des *threads*, et on parle de système multitâche.
Un processus lourd classique, ne contient qu'un seul fil d'exécution, il est **monoprogrammé** :
L'exécution du code du processus est réalisée de manière séquentielle par un fil de contrôle (*thread of control*).



Les threads ou **processus léger** permettent de dérouler plusieurs blocs d'instructions, en PARALLELE, à l'intérieur du même processus. Un thread exécute une fonction. Chaque fonction associée à un thread va être exécutée de façon parallèle et indépendante.




Le système d'exploitation gère donc des centaines de thread :

```
root@beagleboard:~# ps -eo uname,pid,ppid,nlwp,pcpu,pmem,psr,start_time,tty,time
,args
```

id processus fils — id processus pere — Nombre de threads

```
USER        PID   PPID NLWP %CPU %MEM PSR START TT          TIME COMMAND
root          1      0    1  0.6  0.1   0 20:16 ?        00:00:03 init [5]
root          2      0    1  0.0  0.0   0 20:16 ?        00:00:00 [kthreadd]
root          3      2    1  0.0  0.0   0 20:16 ?        00:00:00 [ksoftirqd/0]
root          5      2    1  0.0  0.0   0 20:16 ?        00:00:00 [kworker/u:0]
root          6      2    1  0.0  0.0   0 20:16 ?        00:00:00 [rcu_kthread]
root          7      2    1  0.0  0.0   0 20:16 ?        00:00:00 [cpuset]
```

états du processus — priorité

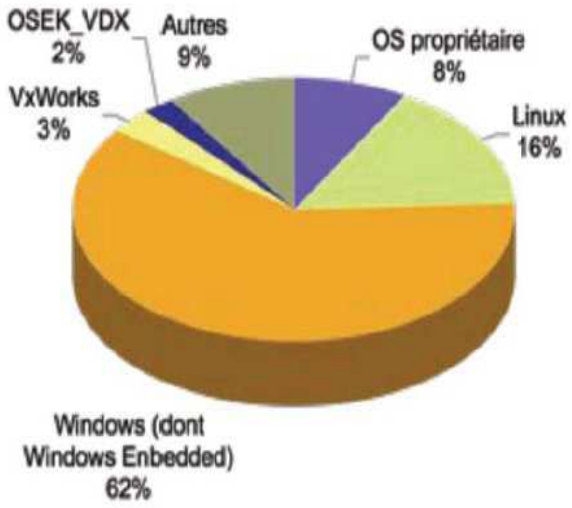
```
root@beagleboard:~# ps -all
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0   876   873  0  80   0 -   748 wait   ttyO2    00:00:00 sh
4 R     0   900   876  0  80   0 -   550 -      ttyO2    00:00:00 ps
root@beagleboard:~#
```

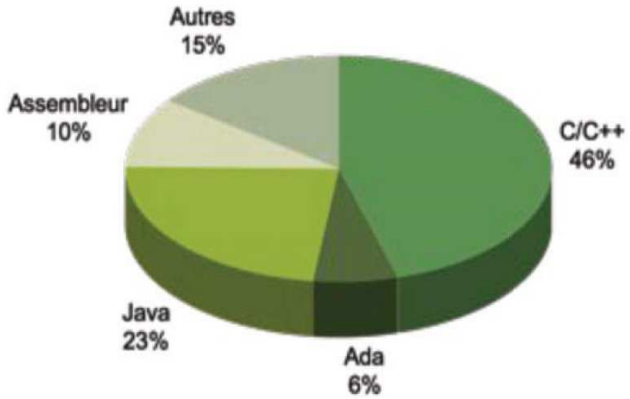
Le processeur lit les instructions des threads à exécuter en mémoire, et une partie d'entre elles agit sur des données en mémoire (lecture/écriture).

Pour faire du multitâche robuste simplement, on est amené à distinguer deux types d'adresse : les adresses pour accéder aux octets en mémoire physique ("adresses physiques"), et les adresses manipulées par le processeur pour ses données et ses instructions ("adresses effectives ou virtuelles" du processeur relatives à l'»espace d'adressage» courant). La traduction adresses effectives adresses physiques est effectuée par un composant particulier, souvent intégré dans le processeur : **la *MMU (memory management unit)*** Cette traduction est réalisée en fonction de la configuration de l'espace d'adressage courant, elle-même stockée en mémoire sous la forme de tables. Les versions courantes du noyau Linux sont prévues pour fonctionner sur des processeurs avec MMU, ce qui concerne la majorité des processeurs utilisés dans la microinformatique classique et aussi dans un bon nombre d'applications embarquées. Un portage du noyau Linux est cependant disponible pour les processeurs dépourvus de MMU : µClinux – pour Micro-C Linux

## 2) Linux et les systèmes embarqués

Un système embarqué peut être *défini* comme un système électronique et informatique autonome, qui est dédié à une tâche bien précise.
II ne possède généralement pas des entrées/sorties standards et classiques comme un clavier ou un écran d'ordinateur. Le système matériel et l'application sont intimement liés, le logiciel embarqué étant enfoui, noyé dans le matériel. Le matériel et le logiciel ne sont pas aussi facilement discernables comme dans un environnement de travail classique de type ordinateur PC. On donne ci-dessous un bilan des systèmes embarqué (OS et langage de programmation):




Windows est souvent cité dans l'embarqué, mais davantage comme OS générique de développement

### 3) Linux et le temps réel

Travailler en temps réel sur un système informatique correspond à élaborer, à partir d'informations acquises d'un phénomène extérieur, des informations de commande ou de contrôle dans un temps cohérent avec l'évolution du phénomène contrôlé.

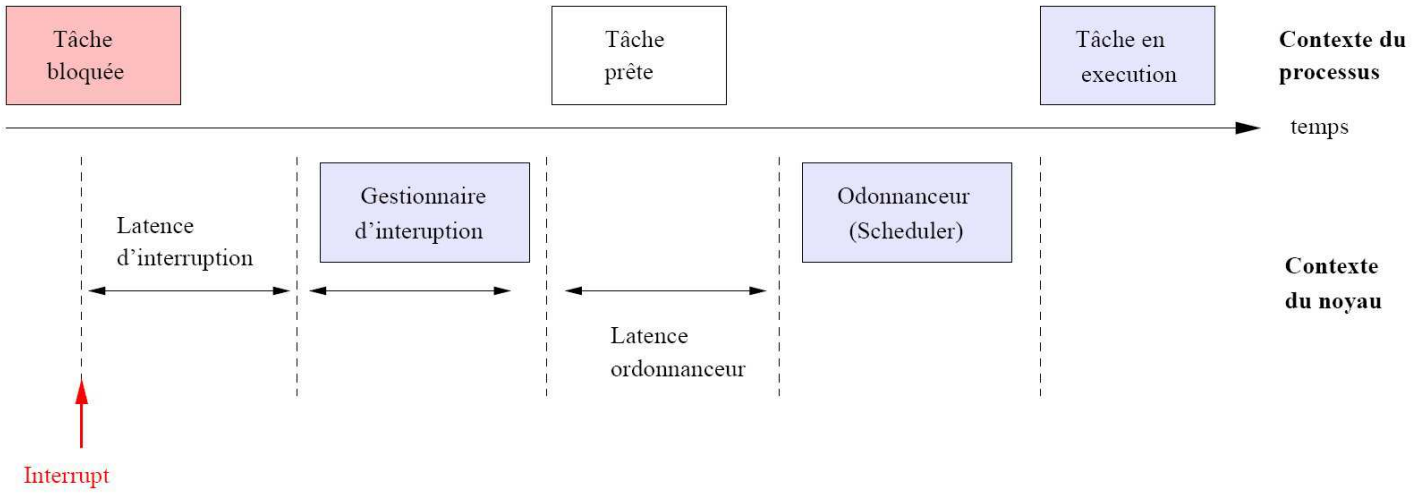
Linux fut développé sur les concepts d'UNIX, ce n'est donc pas un système temps réel, contrairement à la plupart des systèmes embarqués propriétaires comme VxWorks, QNX ou LynxOS.
Au niveau de Linux, il existe divers composants permettant de mettre en place des systèmes temps réel, et ce, à plusieurs niveaux de contraintes (temps réel mou ou dur).
Depuis le noyau Linux 2.6, on a inclus un nouvel ordonnanceur permettant d'améliorer les temps de latence du système.

**Comparaison des temps de réactivité** (source : [2])

- temps de réponse entre l'INT et le démarrage de sa tâche
  (latence d'INT (qq $\mu s$) +durée ISR + latence réordonnanceur + ordonnancement)

| | | |
|---|---|---|
| Linux Standard | non TR | 10-100ms |
| CONFIG_PREEMPT_X | TR mou | qq 100 $\mu s$ |
| CONFIG_PREEMPT_RT | | 100 $\mu s$ |
| Micro noyau RT | TR dur | $< 20\ \mu s$ |

Le temps de Latence est le temps qui sépare la génération d'une interruption du lancement de la tâche d'interruption :

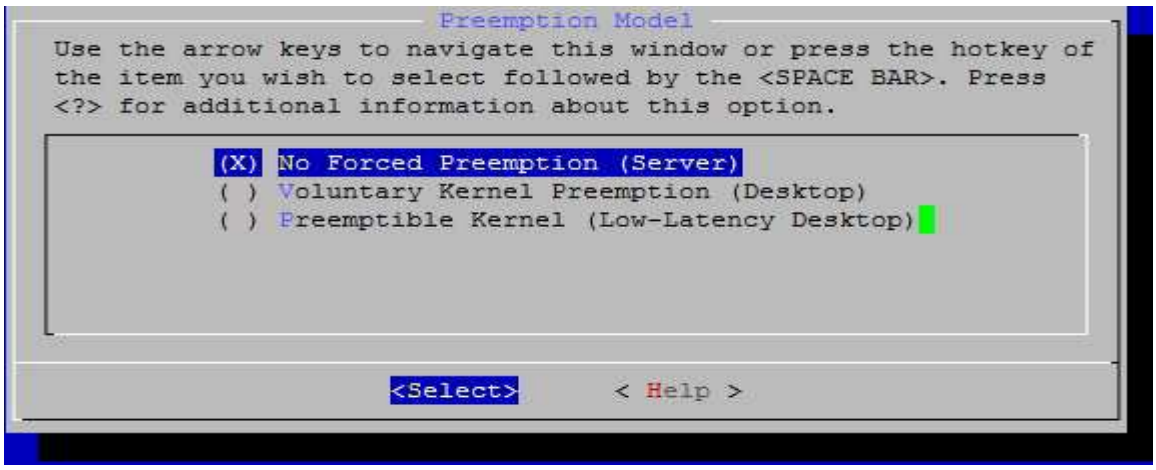



Les noyaux se divisent en deux catégories appelées *noyaux sans réquisition du processeur* (no preemptive scheduling) et ***noyaux avec réquisition du processeur*** (preemptivescheduling).

Dans un **noyau sans réquisition du processeur**, la tâche est exécutée jusqu'à ce qu'elle fasse **appel à un service** du noyau ou **qu'elle s'arrête** d'elle-même. Selon la situation alors présente, le scheduler décide si la tâche doit se poursuivre ou non. On peut donc trouver une configuration avec une tâche prioritaire prête et une moins prioritaire en exécution. Ce noyau est plus simple à écrire.
Dans un ***noyau avec réquisition*** du processeur, une tâche peut à tout instant **perdre le contrôle du processeur** au profit d'une tâche de priorité supérieure. La tâche qui perd le processeur n'a aucune possibilité de le savoir. Ce sont les événements temps réel qui font que l'ordonnanceur peut prendre cette décision à tout instant. Seuls les noyaux de ce type pourront être considérés comme véritablement temps réel.

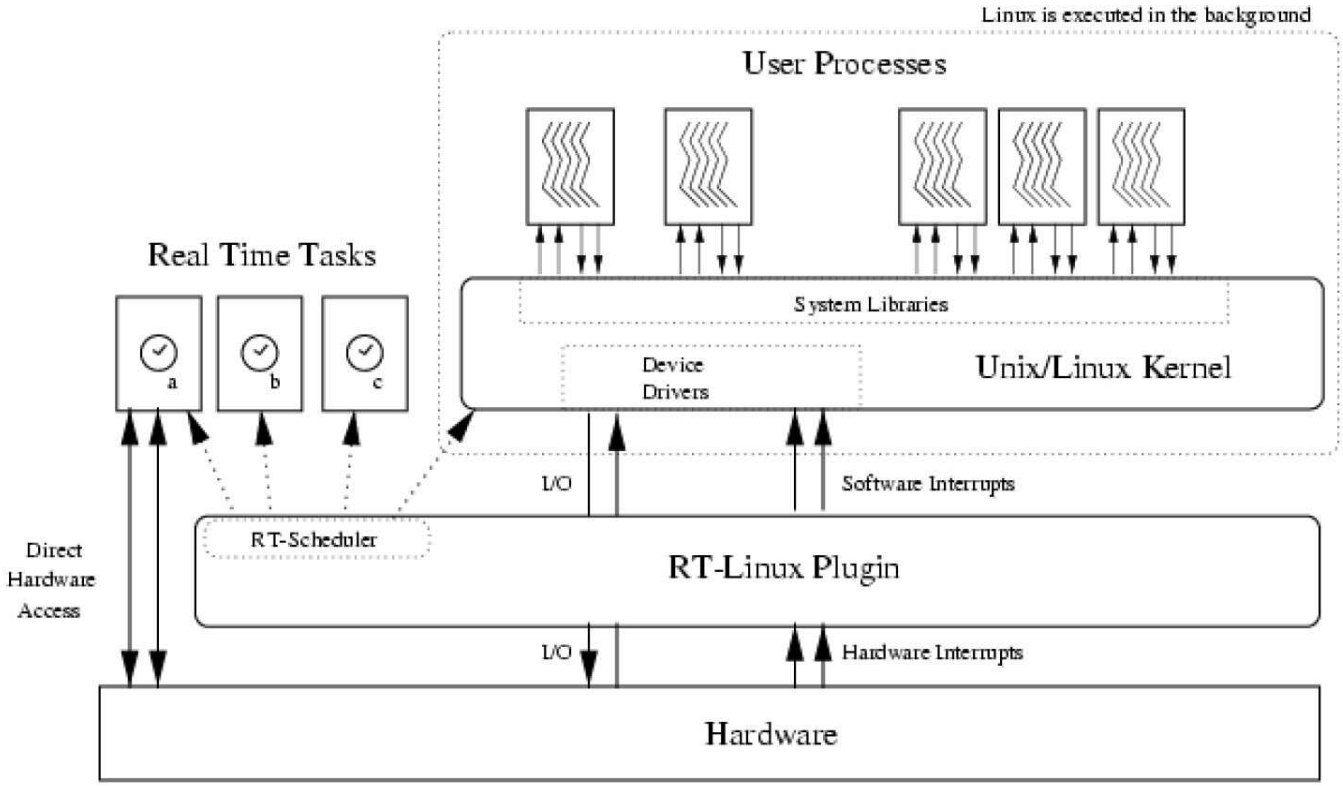
**préemption**: interruption d'une tâche pour une autre plus urgente.

La configuration est accessible dans le menu Processor types and features/ Preemption Model de la configuration du noyau.




Un tel noyau permet d'obtenir des performances satisfaisantes dans le cas d'applications de temps réel mou.
Dans le cas de temps réel dur, il est nécessaire d'utiliser des extensions. Concrètement, cela correspond à un noyau temps réel auxiliaire chargé dans l'espace du noyau Linux sous forme de modules.

Un micro-kernel RT-LINUX temps réel est inséré entre le hardware et Linux. Linux est une tâche de priorité restreinte. Le micro-kernel capture les interruptions pour ses routines avant Linux qui reçoit alors des interruptions virtuelles (Virtualisation des interruptions)




**Les avantages :**

- Le micro-noyau garantie des temps de commutation de contexte très courts performances (< 20µs)
- On bénéficie de tout l'environnement Linux (Xwindows. réseau) qui peut communiquer avec le micro-noyau via des IPC spécifiques

**Les inconvénients :**

- programmation mode kernel –
- pas de protection de l'intégrité du système
- API limitée et simple comparée à des systèmes propriétaires,
- debug parfois complexe.

Le composant Xenomai 2, dérivé du projet RTAI permet de développer des programmes temps réel dur dans l'espace utilisateur (et non dans l'espace noyau), ce qui est un avantage notoire.

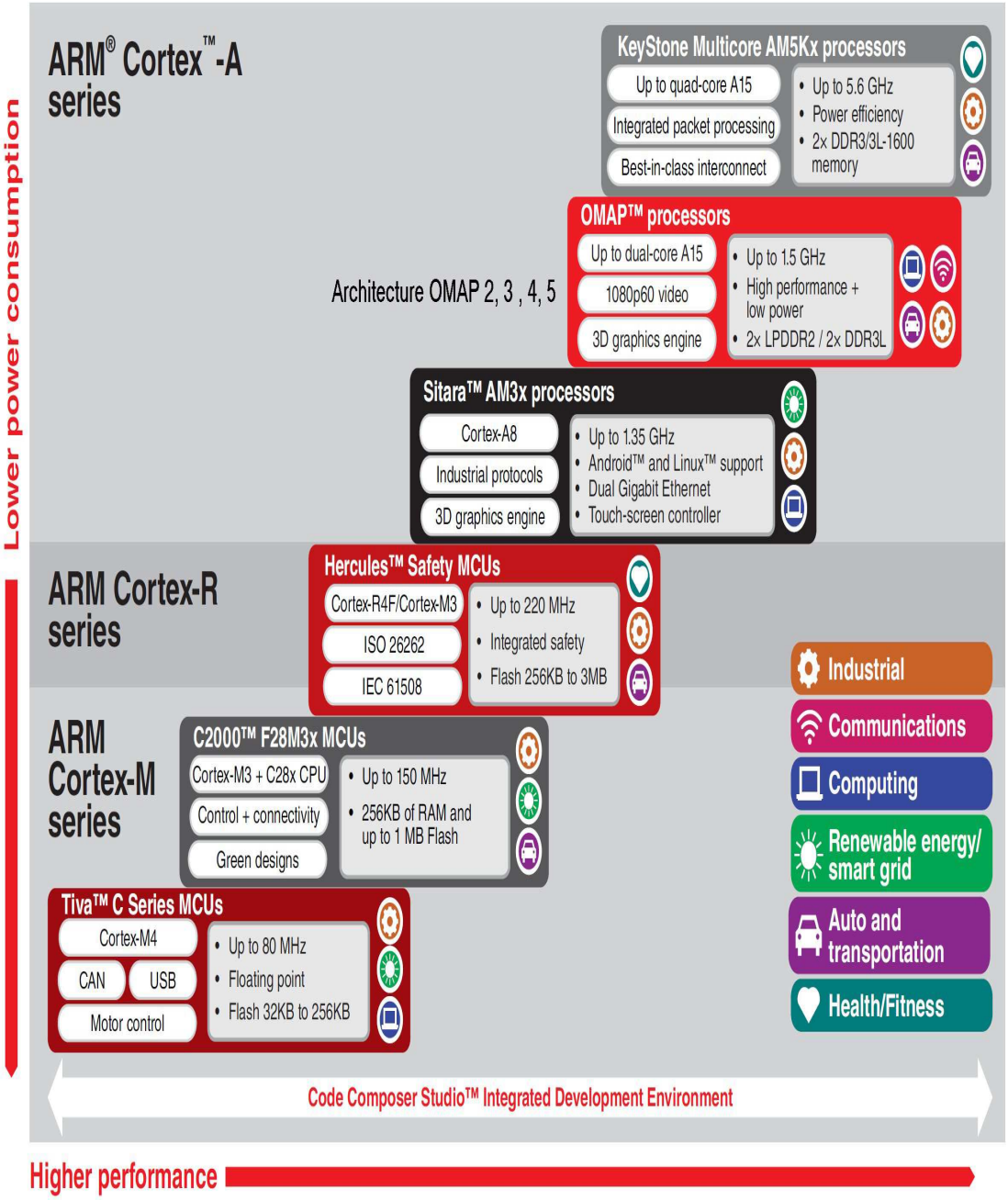
Le noyau Linux standard est initialement conçu pour les processeurs de la famille x86. Depuis la version courante du noyau (2.6) celui-ci contient le portage pour les processeurs suivants :alpha,ARM, H8300, IA64, m68k, m68knommu, MIPS, PowerPC, S390, SH, SH64, SPARC, SPARC64,V850, x86_64... Mais l'on doit souvent patcher les sources du noyau Linux officiel pour tenir compte des particularités spécifiques à un type de processeur dans une famille donnée. II vaut mieux alors s'orienter vers des noyaux patchés, adaptés à son type de processeur et prêts à l'emploi.

| Besoin | Petit | Moyen | Haut de gamme | Embarqué haute disponibilité |
|---|---|---|---|---|
| Taille RAM | <128MO | 128MO | 512MO à 1GO | >4GO |
| Taille ROM/FLASH | <1GO | <4GO | 16 à 32GO | Go-To Disque Dur |
| Processeurs | DragonBall 68k<br>ColdFire<br>ARM | MIPS<br>ColdFire<br>ARM<br>SH<br>PowerPC<br>x86 | | PowerPC<br>x86 |
| Caractéristiques matérielles | Pas de MMU | System on Chip SoC | | |
| Exemples | PDA, téléphone | Routeur, Décodeur, imprimante | | Commutateur téléphonique<br>Serveur haute performance |

L'environnement graphique X Window System (X11) domine depuis longtemps le marché des interfaces graphiques sous UNIX. Ce choix n'est pas forcément le plus judicieux car X11 est complexe, lourd et souvent sous-utilisé dans le cas des applications embarquées. Depuis quelque temps l'utilisation du frame buffer
Linux se répand de plus en plus. Ce mode de fonctionnement correspond au pilotage du contrôleur graphique directement à partir du noyau Linux (fichier spécial 'dev/fb0).
II existe aujourd'hui plusieurs bibliothèques graphiques de haut niveau pouvant utiliser le frame buffer comme alternative à la couche X11. Cela permet de construire des applications graphiques portables, plus facile à mettre au point tout en restant compatible avec les contraintes de Linux embarqué.

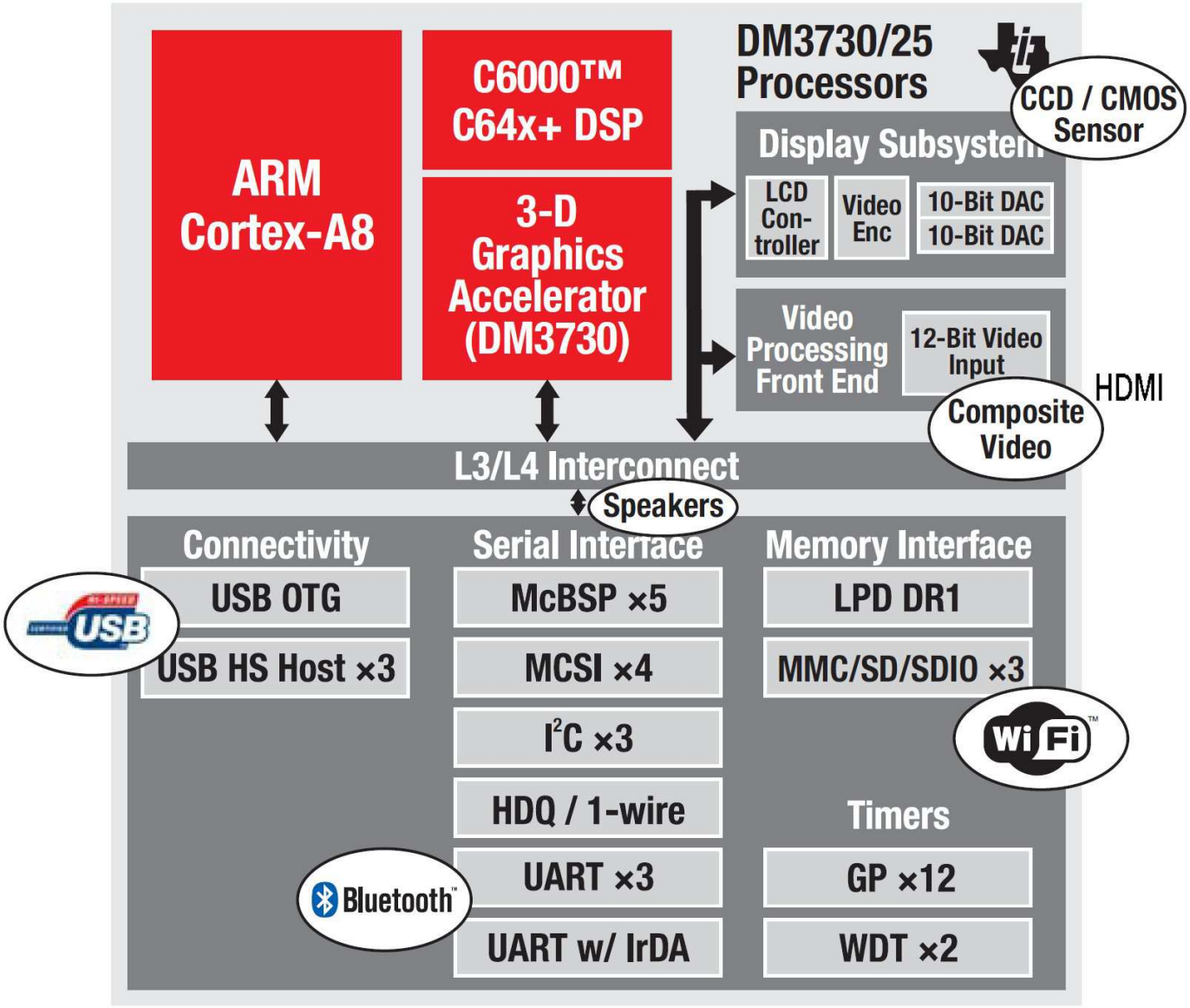
## 4) Architecture de la beagleboard XM

La beagleboard XM est une carte CPU autonome à faible cout pour des systèmes embarqués. Le processeur utilisé sur la beagleboard XM est compatible avec les processeurs ARM cortex A8 fabriqués par Texas Instruments. Texas Instruments propose une gamme importante de microprocesseur répondant à une multitude d'application.




Le processeur DM3730 sur la carte Beagleboard XM est un dérivé de l'architecture OMAP3 de chez Texas Instruments. OMAP est une famille de systèmes sur puces et basée sur une architecture ARM à faible consommation orientée notamment vers les applications multimédias pour systèmes embarquées portables et mobiles.

L'architecture du DM3730 est donnée ci-dessous :

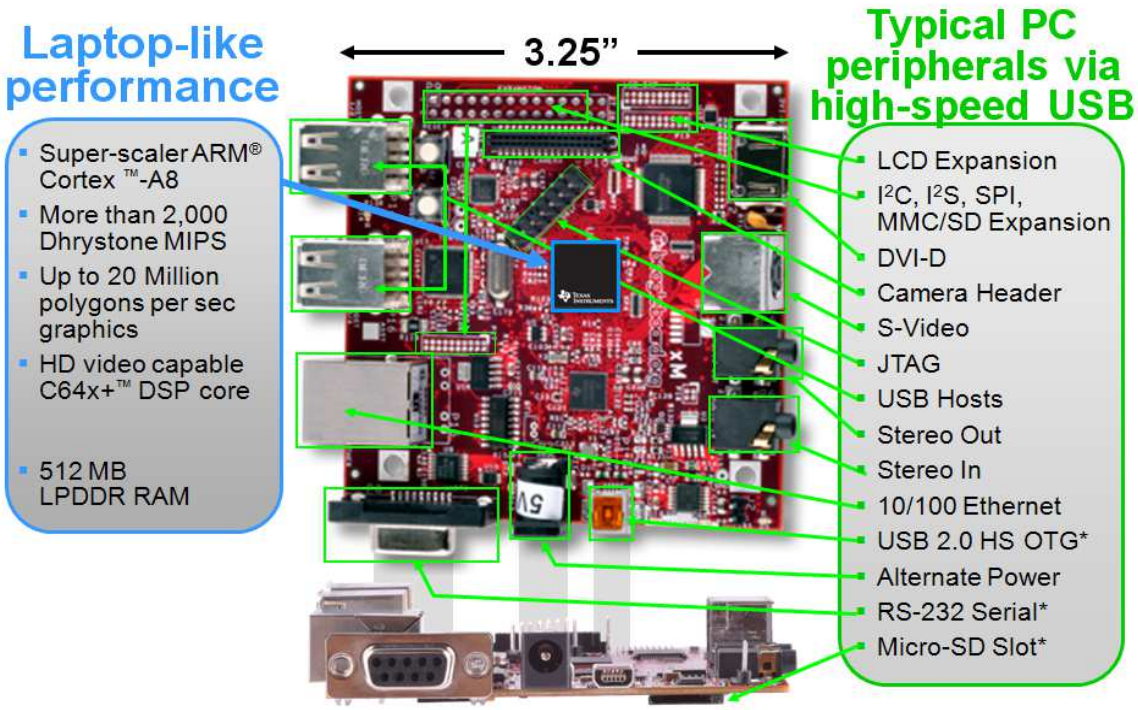



Le **DM3730** est un processeur à base d'un ARM CORTEX A8 , un cœur de DSP à virgule fixe TMS320C64X,un processeur graphique GPU et des périphériques (USB 2.0, SD/MMC…). Tous ces éléments sont intégrés sur la même puce et conviennent pour une multitude d'application dont du traitement vidéo HD. Ces applications peuvent être :

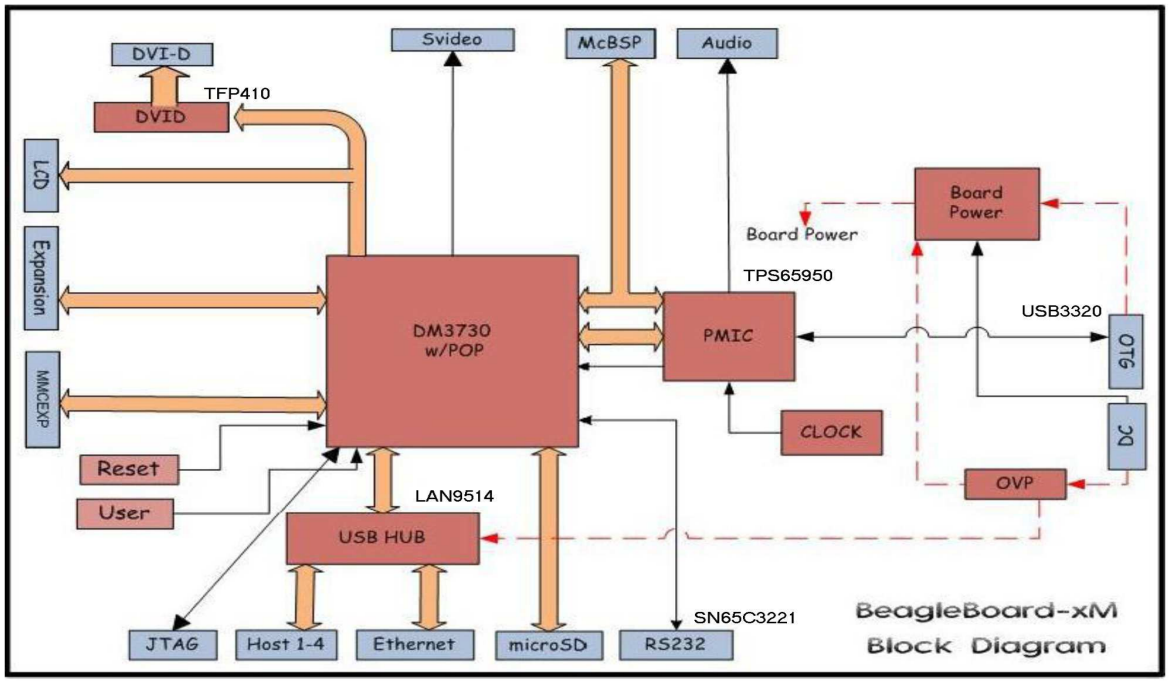
- Systèmes de navigation
- Lecteur multimédia
- Systèmes de surveillance médicale
- Vision industriel
- Communication portable
- Tablette graphique

Attention le bluetooth, l'ethernet, l'USB, le WIFI ne sont pas intégré directement dans le processeur mais le processeur est éventuellement capable de piloter ce type de contrôleur de communication

L'architecture de la beagleboard XM




On retrouve une architecture plus détaillé avec les contrôleurs externes :




Cette carte peut accueillir un système d'exploitation type Linux embarqué, QNX, Android …