Architectures logicielles
Pour systèmes temps réel embarqués

INF342

Plan

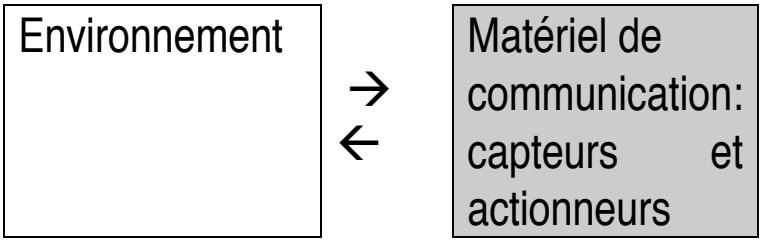
1. Architectures embarquées : de l'application au matériel
    a. Les 3 niveaux : application, support d'exécution, matériel
    b. Les 3 contraintes, conséquences
    c. L'architecture logicielle :
        i. un système réactif
        ii. L'architecture logicielle : adaptation des modèles
    d. Les domaines d'application
2. Les supports d'exécution
    a. Notions de base : priorité, préemption
    b. Evolution
    c. Les différents modèles
    d. Quelques points techniques
3. Les supports d'exécution disponibles
    a. Exemples
    b. Critères de choix
4. Perspectives

Plan

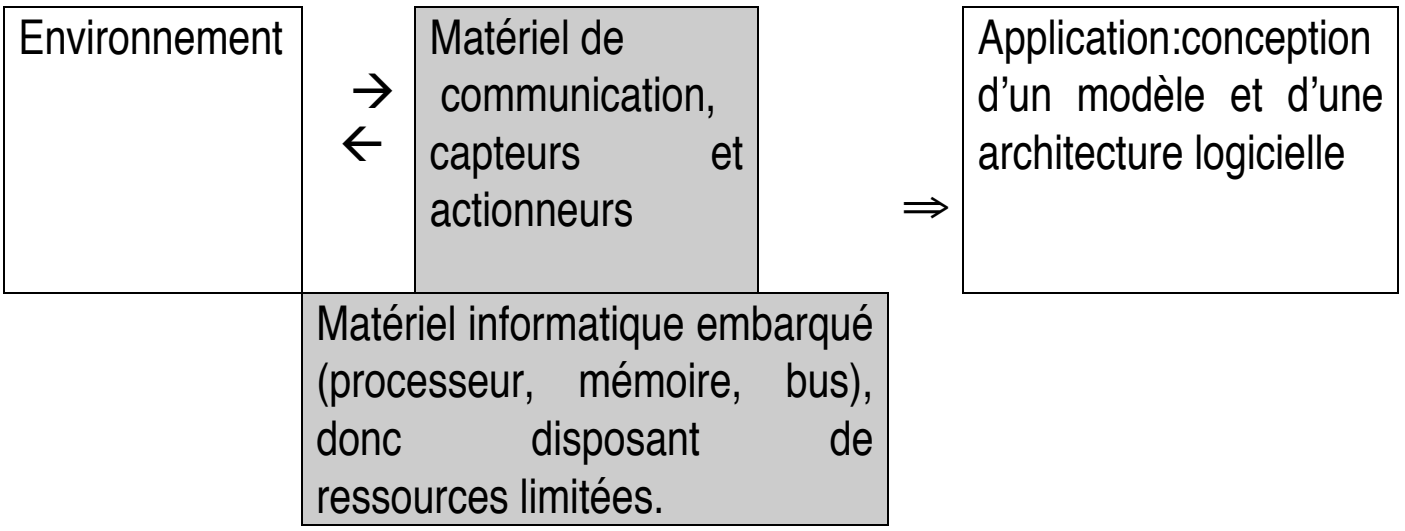
1. Architectures embarquées : de l'application au matériel
    a. **Les 3 niveaux : application, support d'exécution, matériel**
    b. **Les 3 contraintes, conséquences**
    c. **L'architecture logicielle :**
        i. un système réactif
        ii. L'architecture logicielle : adaptation des modèles
    d. Les domaines d'application
2. Les supports d'exécution
    a. Notions de base : priorité, préemption
    b. Evolution
    c. Les différents modèles
    d. Quelques points techniques
3. Les supports d'exécution disponibles
    a. Exemples
    b. Critères de choix

Architecture, partie logicielle : l'application

• Le matériel (robot, …) interagit avec le monde extérieur en utilisant des capteurs et des actionneurs :



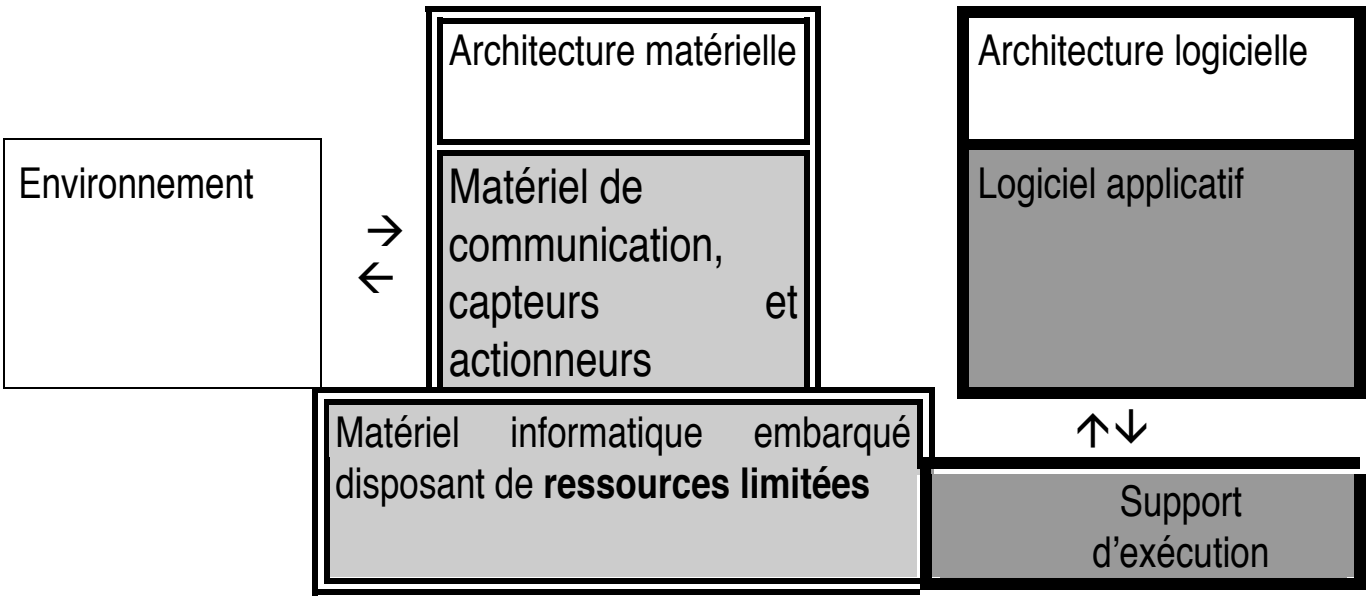
• Pour commander les périphériques, on leur associe une informatique embarquée :



Contrainte 1 : comme toute application TR, l'application est soumise à **des contraintes temporelles**, contraintes imposées par des stimuli **externes** : ceux des périphériques (périodes, échéances, gigues),

Architecture, partie logicielle : support d'exécution

• On choisit, ou on construit, un support d'exécution (interface avec le matériel) pour rendre accessible au logiciel applicatif l'ensemble des ressources matérielles :



• Le support d'exécution peut être rudimentaire (simple gestion des timers et des interruptions) ou très élaboré et fournir des services de haut niveau (synchronisation, ordonnancement, etc)

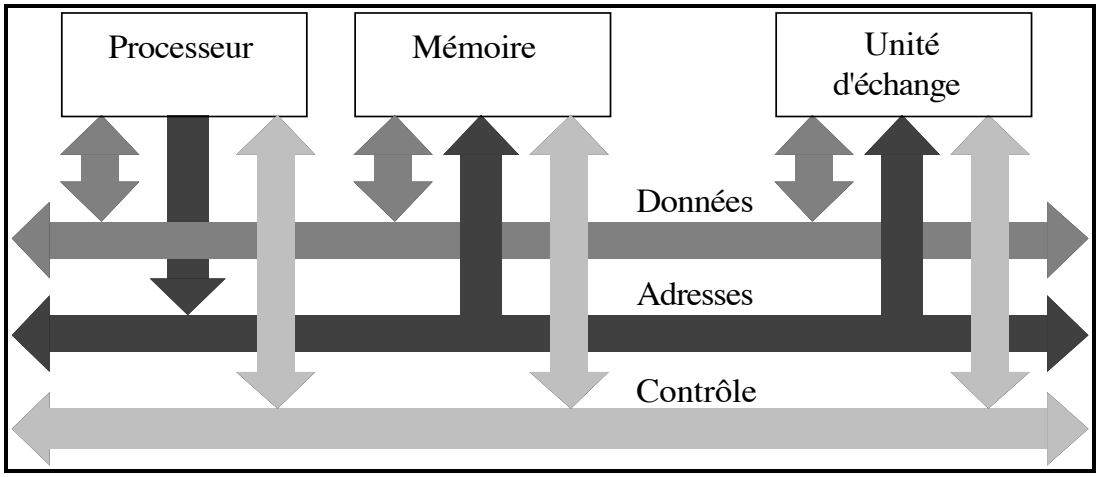
Contraintes 2 et 3 :

- L'architecture logicielle est **contrainte par les ressources matérielles,**

- L'ensemble matériel et logiciel doit garantir : **fiabilité et disponibilité** (sûreté de fonctionnement),

Architecture, partie matérielle

- Schéma simplifié d'une architecture matérielle embarquée :
  - processeur (grandes capacités d'E/S, mais limité en énergie),
  - mémoire,
  - les unités d'échange contrôlent les périphériques, **nombreux et spécifiques** gérés en espace d'adressage unique, ou non,
    - en général : **pas de support de stockage permanent,**



- **Les interactions avec les périphériques sont de deux types** (on peut avoir les combinaisons -1-, -2- , -1et 2-- :
  1. contrôlées par une horloge (échantillonnage, prise de mesures), c'est à dire *time driven* (ou *time triggered)*,
  2. événementielles (alarmes), c'est à dire *event driven* (ou *event triggered)*,

Les trois contraintes : exemples

1. Ressources limitées, c'est à dire : faible poids, faible consommation, faible encombrement. Quelques exemples :
    - mémoire : faible espace d'adressage (*small footprint*,) ⇒ allocation statique,
    - faible réserve d'énergie, or les E/S sont très consommatrices ⇒ la fiabilité, la disponibilité pour éviter des E/S répétées,
    - stockage de masse : souvent absent, *disk on RAM*,
2. Contraintes temporelles :
    - échéances, gigue, WCET
3. Sureté de fonctionnement :
    - la panne d'un composant ne doit pas remettre en cause la vie du système,
    - robustesse : résistance aux vibrations, chocs, température, …

Contraintes : conséquences

• Quelques exemples :

• Niveau applicatif : les algorithmes utilisés dans les STR ne peuvent pas toujours être implantés à cause d'une trop grande consommation de ressources :

- taille mémoire du code, des données,
- dégradation des performances en fonction des restrictions de ressources,

• Niveau support d'exécution : gestion de la mémoire :

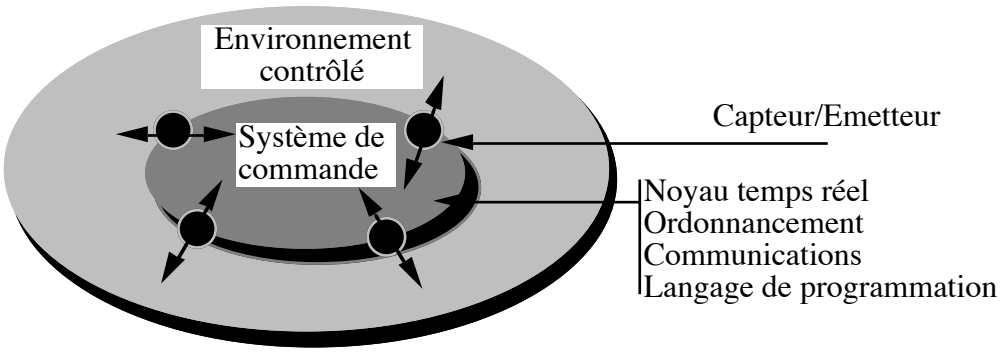
- Les supports d'exécution peuvent proposer des facilités pour dimensionner précisément l'espace d'adressage d'une application,
- Les processus d'allocation dynamique et de récupération sont à revoir (cf. le *garbage collector* des JVM temps réel),

• Gestion des pannes : les interventions de maintenance sont difficiles ou impossibles, ce qui implique :

- un haut niveau de **disponibilité**,
- des possibilités de fonctionnement en mode dégradé,

L'architecture logicielle : un système réactif

• L'architecture logicielle est le système de commande de l'ensemble du dispositif :



• Rôle et mode de fonctionnement de l'architecture logicielle :

- répondre aux stimuli du monde extérieur, en exécutant des traitements soumis à des contraintes temporelles également fournies par l'environnement,
- elle ne décide pas de la base de temps, comme dans les cas des systèmes temps partagé, c'est un système réactif : les contraintes temporelles sont imposées par l'environnement,

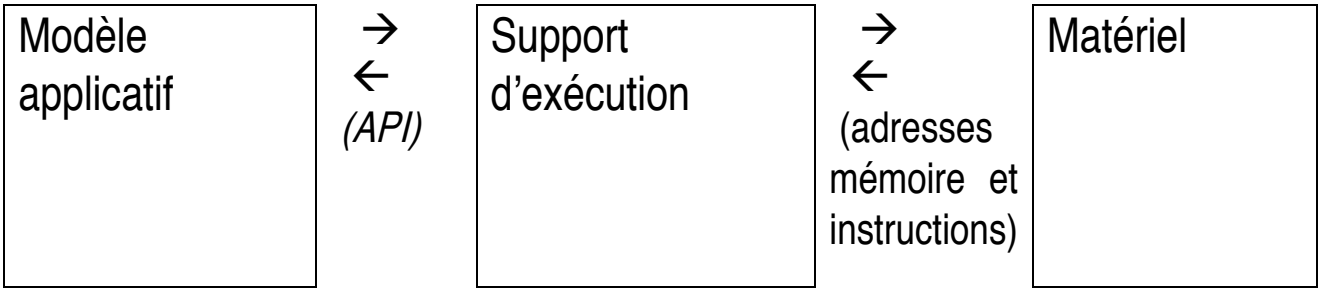
• Un système réactif est en interaction constante avec son environnement, il en reçoit des signaux et répond par des signaux en sortie en exécutant des fonctions soumises à des contraintes temporelles fortes

Plan

1. Architectures embarquées : de l'application au matériel
    a. Les 3 niveaux : application, support d'exécution, matériel
    b. Les 3 contraintes, conséquences
    c. L'architecture logicielle :
        i. un système réactif
        ii. L'architecture logicielle : adaptation des modèles
    d. Les domaines d'application
2. Les supports d'exécution
    a. Notions de base : priorité, préemption
    b. Evolution
    c. Les différents modèles
    d. Quelques points techniques
3. Les supports d'exécution disponibles
    a. Exemples
    b. Critères de choix

Adaptation du modèle du logiciel applicatif A celui du support d'exécution

- L'application interagit avec le monde extérieur en utilisant le support d'exécution qui lui donne accès au matériel :



- Le support d'exécution peut être plus ou moins complexe et son modèle de tâches (ordonnancement) plus ou moins proche de celui de l'application,

- Problème : les styles de fonctionnement de ces trois composantes peuvent donc **différer**, par exemple :
  - modèle de tâche applicatif : RMS, ordonnancement fourni par les support d'exécution : FCFS (FIFO) préemptif, matériel event triggered,
    - conséquence : l'application devra donc implémenter RMS en utilisant FIFO, et le support se satisfaire d'un seul type de trigger,

Adaptation du modèle du logiciel applicatif A celui du support d'exécution

• Exemples de problèmes d'ajustement des différents modèles :

Modèle applicatif	RMS		
Support d'exécution	RTEMS (FIFO préemptif)	Linux RT	ARINC
Matériel	Type des triggers?		
	⇓	⇓	⇓
Problème :	Comment gérer L'adaptation RMS sur FIFO ?		

- Les timers matériels sont plus ou moins précis :
    - l'architecture logicielle doit prendre en compte la gestion des gigue et des dérives d'horloges,

Plan

1. Architectures embarquées : de l'application au matériel
    a. Les 3 niveaux : application, support d'exécution, matériel
    b. Les 3 contraintes, conséquences
    c. L'architecture logicielle :
        i. un système réactif
        ii. L'architecture logicielle : adaptation des modèles
    d. Les domaines d'application
2. Les supports d'exécution
    a. Notions de base : priorité, préemption
    b. Evolution
    c. Les différents modèles
    d. Quelques points techniques
3. Les supports d'exécution disponibles
    a. Exemples
    b. Critères de choix

Domaines d'application
Des architectures logicielles TR embarquées

• Les domaines traditionnels :

- spatial, avionique (systèmes temps réel répartis embarqués),
- robotique,
- contrôle de processus industriels (chimie, nucléaire, …),

• Les nouveaux domaines:

- transport,
- médical,
- électronique grand public (jeux), téléphonie mobile,
- systèmes embarqués temps réel répartis à grande échelle (*digital city*,…),
- informatique diffuse (*pervasive computing, ubiquitous computing*), *wearable computer*, domotique, objets intelligents (*smart objects*), immeubles intelligents…

Le marché

• Tendances :

- Ce marché devient un marché de masse,
- Standardisation (POSIX temps réel, embedded Linux, RTSJ),

• **Marché atomisé** :

- Pas d'offre globale (on compose soi-même sa configuration tant au niveau matériel que logiciel),
- Pas de fournisseur dominant (31% de systèmes dits "propriétaires"), Microsoft, Sun, IBM sont marginaux,

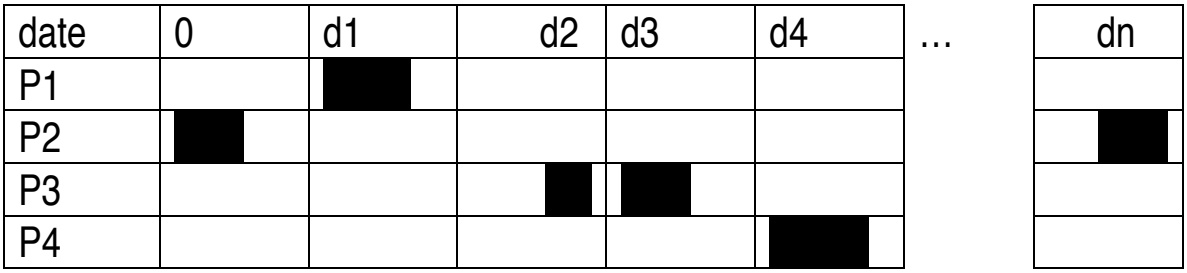
Secteur	Maturité	Contraintes	Tendances
Aérospatilae, militaire	Forte	Performance, fiabilité	Réduction des coûts (économie d'échelle)
Transports	Moyenne	Fiabilité, coût	Ergonomie, réseau
Industrie	Forte	Fiabilité, coût	Réseau
Télecomm.	Forte	Performance, Fiabilité	Puissance
Grand public	Faible	Performance, coût	Internet
Commerce elec.	Faible	Fiabilité, coût, sécurité	Commerce en ligne

Plan

1. Architectures embarquées : de l'application au matériel
    a. Les 3 niveaux : application, support d'exécution, matériel
    b. Les 3 contraintes, conséquences
    c. L'architecture logicielle :
        i. un système réactif
        ii. L'architecture logicielle : adaptation des modèles
    d. Les domaines d'application
2. **Les supports d'exécution**
    a. Notions de base : priorité, préemption
    b. Evolution des supports d'exécution
    c. Les différents modèles
    d. Quelques points techniques
3. Les supports d'exécution disponibles
    a. Exemples
    b. Critères de choix

Retour sur le modèle applicatif

- L'application interagit avec l'extérieur en dialoguant avec les périphériques. On donne ci-dessous le diagramme d'activité des 4 tâches gérant les périphériques P1 à P4.



- On a donc plusieurs activités en parallèle, un seul processeur : il va falloir gérer une file d'attente pour l'accès à ce processeur, c'est à dire ordonnancer ces activités (ordonnancement hors-ligne possible) :

FA des tâches	date	0	t1	t2	T3	…	tn
	Tâche	P2	P1	P3	P4		P2

- Si des interruptions surviennent pendant le déroulement d'une activité, il va falloir gérer deux files d'attente :

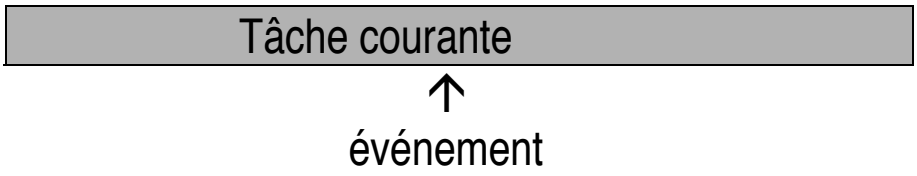
FA des tâches	0	t1	t2	T3	…	tn
	P2	P1	P3	P4		P2

FA des interr.	0	t1	t2	T3	…	tn
	P2	P1	P3	P4		P2

Support d'exécution : gestion des évènements

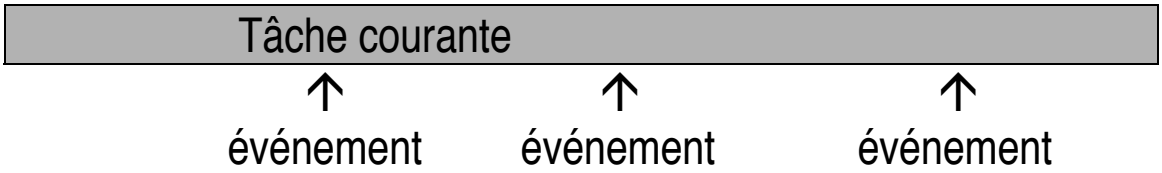
• Comment gérer l'occurrence d'événements pendant l'exécution d'une tâche ?

Cas 1, occurrence d'un seul événement pendant l'activation d'une tâche :



Question : continuer la tâche courante ou exécuter le traitement correspondant à l'événement ?
Réponse : **préemption**.

Cas 2, occurrence de plusieurs événements pendant l'activation d'une tâche :



Question : continuer la tâche courante ou exécuter le traitement correspondant à l'un des événement, et, si oui ; lequel ?
Réponse : **priorité**.

Le concept de préemption

• Que faire pour traiter l'événement si un trigger se déclenche au cours d'une exécution : attendre la fin de l'activité courante ou le traiter immédiatement ? Réponse, préemption ; c'est à dire :

1. arrêt de la tâche courante,
2. sauvegarde du **contexte** de cette tâche courante,
3. si l'événement externe (ré)active une tâche plus prioritaire que la courante, alors :
    a. créer ou restaurer le **contexte** de la tâche à démarrer ou à réactiver,
    b. sinon, restaurer le contexte de la tâche courante

• Problèmes liés :

- quelles tâches peut-on préempter (notion de **priorité des évènements**) ?
- gestion fine du contexte,
- gestion des données partagées :
    - entre tâches, entre tâches et interruptions,

Gestion des priorités

- L'attribution de priorités aux activités permet de résoudre le problème suivant :
    - lorsque plusieurs tâches sont exécutables (c'est à dire se trouvant dans l'état prêt), laquelle d'entre elles doit être activée ?
    - réponse : la plus prioritaire, **mais…**

- **… comment** attribuer les priorités :
    - priorité fixe (RMS) ou variable (EDF),
    - la priorité doit-elle refléter :
        - le respect des échéances,
        - la criticité (la tâche dont l'exécution est la plus importante n'est pas forcément celle dont l'échéance est la plus proche).

Plan

1. Architectures embarquées : de l'application au matériel
    a. Les 3 niveaux : application, support d'exécution, matériel
    b. Les 3 contraintes, conséquences
    c. L'architecture logicielle :
        i. un système réactif
        ii. L'architecture logicielle : adaptation des modèles
    d. Les domaines d'application
2. **Les supports d'exécution**
    a. Notions de base : priorité, préemption
    b. Evolution
    c. Les différents modèles
    d. Quelques points techniques
3. Les supports d'exécution disponibles
    a. Exemples
    b. Critères de choix

Evolution des supports d'exécution (1)

Structure d'une architecture logicielle embarquée dans les années 1970 :

1- Logiciel applicatif :

Application souvent écrite à la fois en langage machine et en langage de haut niveau

2- Logiciel de base (support d'exécution) :

Souvent une simple de boucle : tests de variables d'état associées à chacun des capteurs et exécution de fonctions appropriées.
Mise à jour des données et des registres par les procédures de gestion des interruptions à bas niveau (ISR, Interrupt Service Routines). Ces procédures sont écrites en assembleur. **Le support d'exécution (minimaliste et efficace) est très adapté à une plate-forme matérielle. Portabilité ? Réutilisation ?**

Note :

Ce type de support d'exécution, très peu consommateur de ressources, convient si l'application et le matériel présentent peu de variabilité.

Evolution des supports d'exécution (2)

Structure d'une architecture logicielle embarquée actuelle:

1- Logiciel applicatif :

Application souvent écrite en langage de haut niveau (Java RTSJ, Ada, …)

2- Logiciel de base (support d'exécution) :

Bibliothèques (exemple : *POSIX threads)*
Intergiciel (exemple : *RT CORBA*)
Noyau temps réel embarqué (exemple : RTEMS) qui offre : o Ordonnancement, synchronisation, gestion mémoire (indépendants de la configuration matérielle) o **Support de différents BSP (ou HAL) : portabilité, réutilisation**
Partie spécifique à un ensemble (processeur et carte électronique spécifique) : ***BSP, Board Support Package ; HAL, Hardware Layer***

Evolution des supports d'exécution (3)

• Cette structuration en nombreuses couches répond à :

- la grande diversité des plateformes matérielle :
  - ajout/retrait des capteurs/émetteurs, le *Hardware Layer* ou le *Board Support Package* sont une couche qui sépare du noyau les sections de code spécifiques au processeur et aux périphériques,
  - dérives temporelles des horloges,
- la **variabilité** des applications :
  - nombre de tâches, différence WCET/temps effectif,

Evolution des supports d'exécution (4)

- Le support d'exécution (embarqué, il doit donc être peu consommateur de ressources **!**) offre donc une variété de fonctionnalités permettant de construire un système **réactif** :

  - Une (ou plusieurs) politique(s) d'ordonnancement,
  - des outils de synchronisation adaptés au temps réel (PCP, PIP),
  - Des moyens de répondre sous contraintes temporelles aux événements externes,
  - Files des messages "temps réel"
  - …

- Nous avons vu que les deux concepts à la base des ces fonctionnalités sont :

  - Préemption,
  - Priorité,

Plan

1. Architectures embarquées : de l'application au matériel
    a. Les 3 niveaux : application, support d'exécution, matériel
    b. Les 3 contraintes, conséquences
    c. L'architecture logicielle :
        i. un système réactif
        ii. L'architecture logicielle : adaptation des modèles
    d. Les domaines d'application
2. **Les supports d'exécution**
    a. Notions de base : priorité, préemption
    b. Evolution
    c. Les différents modèles
    d. Quelques points techniques
3. Les supports d'exécution disponibles
    a. Exemples
    b. Critères de choix

Modèles pour les supports d'exécution

• Modèles sans tâche (prédictibles mais pas flexibles, efficaces : simples appels de fonctions) :

- sans boucle,
- avec boucle,
  - sérialisation des événements

• Modèles avec tâches, on peut distinguer deux familles, en fonction du type de services proposés :

- ordonnanceur avec gestion élémentaire de la concurrence,
  - l'interface avec le matériel se fait directement à bas sniveau. Ce modèle est bien adapté si la configuration matérielle et le modèle applicatif sont très proches,
- noyau modulaire et interface à base de services, notion de HAL,
  - bien adapté si la variabilité est grande, par exemple dans le cas d'une configuration matérielle dotée de capteurs nombreux, variés, et dont les temps de réponse sont très variables.

Modèle sans tâche

• Domaines d'application du modèle :

- fortes contraintes sur les ressources matérielles, donc peu (ou pas) de variabilité,
- peu de parallélisme à gérer, l'ordonnancement hors ligne est possible,
- efficace, difficilement portable, pas réutilisable,

• Implémentation du modèle :

- sans boucle
  - sérialisation (bin packing),
- avec boucle
  - boucle de scrutation des événements,

Modèle sans tâche : La boucle de scrutation des événements

- C'est une suite de tests qui vérifie l'état de chaque périphérique (**sérialisation**) :
  - Si cet état a changé, alors exécution d'une action,
  - Sinon : rien.
- Avantages :
  - Latence du système = temps de traitement d'une boucle,
  - Simple à programmer,
- Inconvénients
  - Alignement de la boucle sur le périphérique le plus lent,
  - Programme difficile à maintenir,
  - La suite définit un ordre, donc une **priorité** implicite, sur les traitements,
  - Pas de gestion immédiate des interruptions : le dialogue avec les périphériques se fait à l'initiative de l'application (alors que l'objectif peut être du type *event driven*),

Modèle avec tâches : Exécutif cyclique« time driven »

• Fonctionnement dérivé de celui de la boucle:

- Un *timer* lance les tâches,
- Chacune est exécutée en totalité avant de passer à la suivante,
- La dernière doit se terminer avant la prochaine interruption *timer,*
- Pour gérer des événements qui ont des fréquences différentes, on peut exécuter certaines tâches plusieurs fois dans un cycle (**cycle majeur et cycle mineur**),

Modèle avec tâches : Exécutif/système

• Exécutif temps réel, moniteur temps réel (par exemple, RTEMS):

- **recompilé** avec l'application, se présente sous forme d'une bibliothèque, « linkée » avec l'application. On recharge l'ensemble [application+système] à chaque fois qu'on relance l'application.
- Il est de très petite taille, parce que **modulable** en fonction de l'application,
- gestion mémoire très simple donc efficace en terme de temps,
- pas d'interface utilisateur, le couple écran/clavier n'est plus le périphérique privilégié,

• Système temps réel (par exemple, Linux) :

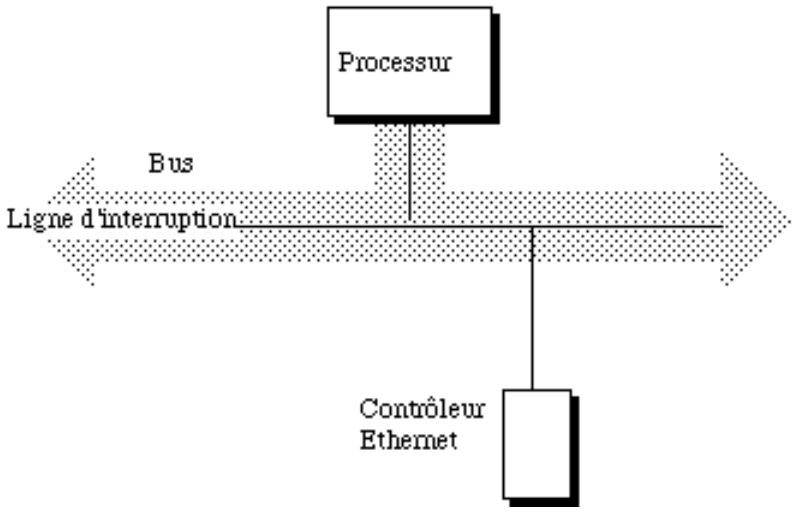
- système chargé en entier en mémoire, il reste en mémoire et permet l'exécution successive de plusieurs applications,
- mais il est plus volumineux qu'un exécutif,
- plus portable (?), souvent compatible POSIX,
- utilisation plus souple (interface du type shell…), mais est-ce toujours nécessaire ?

Plan

1. Architectures embarquées : de l'application au matériel
    a. Les 3 niveaux : application, support d'exécution, matériel
    b. Les 3 contraintes, conséquences
    c. L'architecture logicielle :
        i. un système réactif
        ii. L'architecture logicielle : adaptation des modèles
    d. Les domaines d'application
2. **Les supports d'exécution**
    a. Notions de base : priorité, préemption
    b. Evolution
    c. Les différents modèles
    d. Quelques points techniques
3. Les supports d'exécution disponibles
    a. Exemples
    b. Critères de choix

Les interruptions

• Elles sont l'outil qui permet aux de renvoyer les événements externes vers le processeur, par exemple :



• Mots clés : tables des vecteurs d'interruption, priorités, masquage,

• Horloges, par exemple :

- time of day clock :
  - mise à jour d'un compteur qui est éventuellement lu par le système ou les applications,
- *real time clock* (interrompt le processeur) :
  - envoie une interruption à chaque *tick*,
  - sert à gérer des alarmes (quantum, échéance), utilisation d'une liste du type delta,

Ordonnancement, la « processs table »

• Structure de la table des processus, ou « process table » :

- une entrée par tâche,
- chaque entrée contient (ou permet de retrouver) :
  - **l'état** de la tâche,
  - une copie des valeurs des registres lors de la dernière préemption de cette tâche, en particulier celles du compteur ordinal et du pointeur de pile **(sauvegarde du contexte),**
  - des informations sur la consommation des ressources, par exemple :
    - le nom du sémaphore sur lequel il est bloqué,
    - les messages qui lui ont été envoyés,
    - les périphériques utilisés,

Ordonnancement : mécanisme
Du changement de contexte (1)

• La décision de changement de contexte (passage d'un tâche à une autre) peut intervenir après une préemption, une opération sur un sémaphore, une interruption…

• Cette décision est prise par le scheduler (ordonnanceur) (qui peut être une simple fonction `sched` ():

- il choisit le plus prioritaire des processus se trouvant dans l'état READY (prêt) ou CURRENT (actif, running),
- il fait passer (ou laisse, dans ce cas pas de changement de contexte) le processus choisi dans l'état CURRENT,
- fait éventuellement passer le CURRENT en READY,
- appelle une fonction du type `ctxswitch`,

Le changement de contexte : La fonction où on ne revient jamais…

• La fonction `ctxswitch`:

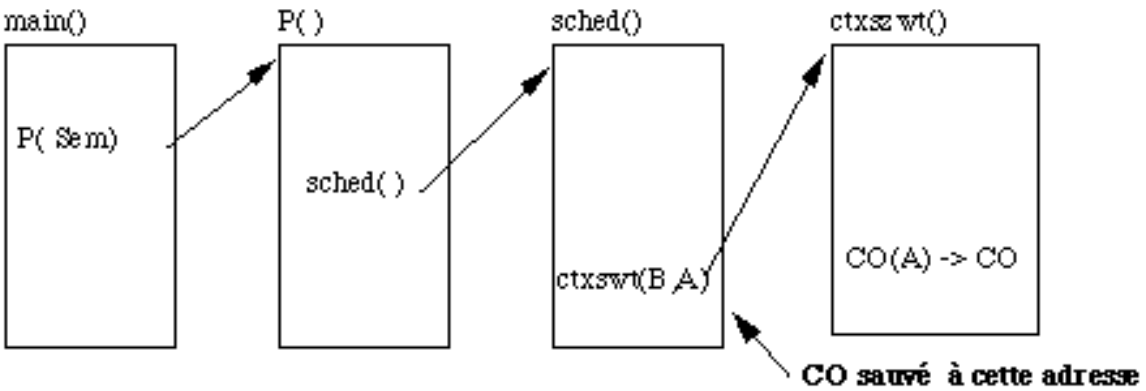
- sauve les registres,
- restaure le contexte du processus réordonnancé,
- ATTENTION : dès que le compteur ordinal est restauré, on part dans le code du processus réordonnancé : on ne revient pas de la fonction `ctxswitch q`ui a fait ce changement de contexte !!!!

• Dans l'exemple suivant :

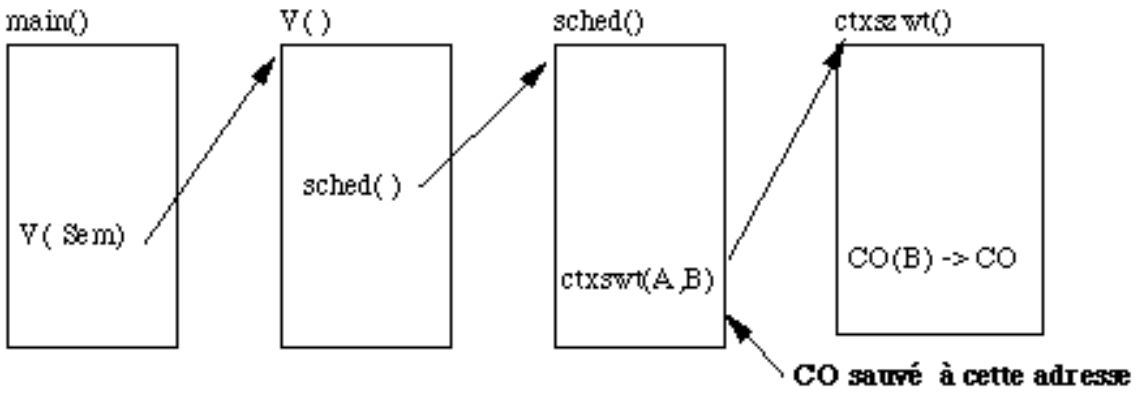
- le processus B se bloque sur une opération P sur un sémaphore Sem,
- P appelle le scheduler qui donne la main au processus A,
- Changement de contexte en faveur de A, **appel à ctxswt**(B,A),
- On ne sort pas de la fonction ctxswt, on retourne dans le programme exécuté par A,
- `A fait V(Sem)`, ce qui va réactiver B, parce que B est plus prioritaire que A, on restaure le contexte de B
- On revient dans le programme exécuté par B,

Le changement de contexte :

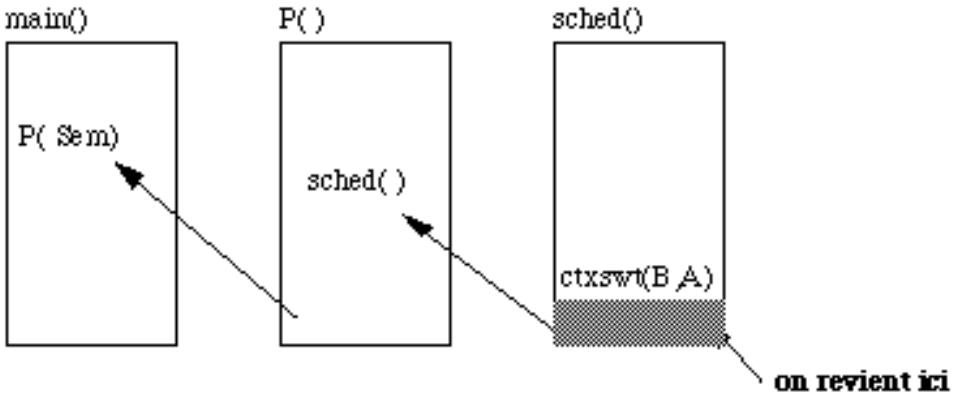
Dans le contexte de B



Dans le contexte de A



Dans le contexte de B, les retours de fonction :



Changement de contexte Et ordonnancement

• Quelques précisions :

- la valeur du compteur ordinal sauvée sur la pile lors de **l'appel** à `ctxswitch` est bien sa propre adresse de retour, comme pour toutes les fonctions :
    - c'est celle où devrait revenir `ctxswitch` si elle était une fonction ordinaire… mais `ctxswitch` sauve le contexte courant du processus A, et restaure la valeur du CO qui avait été sauvée dans le programme exécuté par B, `ctxswitch` n'exécutera donc jamais pas son propre return…, l'exécution continue dans le programme exécuté par B,
- les processus appellent `sched`() pour décider quel processus doit être active, donc :
    - tous les processus suspendus reviennent dans `sched`() après l'appel à `ctxswitch,`
    - c'est le retour de `sched` ()qui les renvoie dans la fonction qui a appelé `sched()`

Le processus null ou idle,

- le rôle du scheduler est seulement de choisir un processus parmi les READY et CURRENT :
  - il ne créé PAS de processus,
  - il ne vérifie pas si la liste des READY est vide,
- Conséquence : il doit toujours y avoir au moins un processus dans l'état READY : le `null process,`

Gestion de la concurrence : Les sémaphores

• Opération P :

- si le compteur passe négatif :
  - faire passer le processus de CURRENT à BLOCKED,
  - indique le numéro du sémaphore dans l'entrée de la process table associée au processus,
  - appel à `sched(),`

• Opération V :

- si le compteur passe négatif ou nul :
  - choisir un processus et le faire passer de BLOCKED à READY,
  - appel à `sched(),`

Les entrées-sorties

• Elément déterminant d'un STRE : la gestion des entrées-sorties :

`- lire (périph. , données, échéance)`

• On doit pouvoir gérer les e/s comme on gère les processus :

- satisfaire la plus prioritaire d'abord,
- annuler une demande lorsque son échéance est dépassée,
- réordonner les demandes si une plus prioritaire arrive,

• Si les tâches partagent des ressources (capteurs/émetteurs, accès au réseau) :

- utiliser des algorithmes d'ordonnancement entre tâches dépendantes,
- utiliser PCP ou PIP pour accéder aux sémaphores gérant les ressources,

• Attention aux mécanismes matériels :

-entrées-sorties exécutées par le processeur,

-entrées-sorties indépendantes du processeur (DMA),

Plan

1. Architectures embarquées : de l'application au matériel
    a. Les 3 niveaux : application, support d'exécution, matériel
    b. L'architecture logicielle : un système réactif
    c. L'architecture logicielle : adaptation des modèles
    d. Les domaines d'application
2. Les supports d'exécution
    a. Notions de base : priorité, préemption
    b. Evolution
    c. Les différents modèles
    d. Quelques points techniques
3. **Les supports d'exécution disponibles**
    a. Exemples
    b. Critères de choix

Quelques supports d'exécution temps réel embarqués

• **Propriétaires :**

- VxWorks, de WindRiver (NASA : Mars Pathfinder, …),
- VRTX, de Mentor Graphics, téléscope Hubble,
- LynxOS de Lynuxworks, armement,

• **free software :**

- RTEMS de OAR, implémenté en C et Ada, porté sur de très nombreux processeurs et BSP,
- Linux embarqués variés (RT-Linux, Linux-RTAI)

• **universitaires :** Spring (EDF), ARTS, MARTE (RT POSIX), TinyOS

• **spécifications :**

- JVM : la version RTSJ,
- ARINC 653, avionique haute disponibilité,

o `http://www.lynuxworks.com`
o `http://marte.unican.es`
o `http://www.tinyos.net`
o `http://www.rtems.com`
o `http://www.arinc.com`

RTEMS

- RTEMS (*Real-Time Executive for Multiprocessor Systems*) de OAR (*On-Line Applications Research Corporation*) http://www.rtems.com :

  - temps réel dur, porté sur de très nombreuses plateformes matérielles,
  - exécution déterministe : elle ne varie pas en fonction du nombre d'objets présents sur le système, temps de latence dû aux interruptions prédictible,
  - ordonnancement préemptif, basé sur les priorités, RMS fourni,
  - le système est configurable : seules sont chargées les fonctionnalités (tâches, sémaphores, timers, …) nécessaires à une application donnée (notion de *manager*), on optimise ainsi les performances en vitesse et on minimise l'espace d'adressage utilisé,
  - possibilité d'intégrer des services spécifiques par la mise en œuvre d'*extensions* (pour insérer des traces, etc)
  - implémenté en C et Ada, API POSIX, versions (BSP) pour Motorola PPC, M68xxx, Intel ix86, i960, MIPS, SPARC, …

Linux temps réel

- RT Linux (Real time Linux), http://www.rtlinuxfree.com et RTAI (Real Time Application Interface for Linux), http://www.rtai.org) :
  - Construits à partir du système temps partagé Linux, qui présente les défauts suivants :
    - sections de code du noyau pendant l'exécution desquelles les événements externes sont masqués,
    - support partiel de l'interface POSIX temps réel.
- Deux techniques pour rendre Linux temps réel :
  - applications de *patches* sur le noyau pour y placer des points de préemption,
  - insertion d'un second noyau entre Linux et le matériel (approche adoptée par RTAI),

Symbian

• Symbian, http://www.symbian.com :

- construit au dessus de EKA2 qui est un noyau mono utilisateur, multitâches, préemptif, peu consommateur de mémoire et d'énergie. Priorités préemptives, synchro. PIP,
- appels système en temps borné, *timer* haute résolution, latence pour le traitement des interruptions est de 0,5ms à 1ms. Symbian ne fait pas d'hypothèse sur le MMU du processeur et propose plusieurs modèles de mémoire.
- outils de communication entre threads, de gestion des interruptions, d'écriture de pilotes, des moyens pour le contrôle gestion de l'énergie et la définition du HAL. La gestion de la sécurité et des fichiers est également prévue.
- Gestion de pile GSM.

ARINC 653

• ARINC 653 (Aeronautical Radio Inc, http://www.arinc.com) est une spécification pour l'avionique qui décrit l'interface que doit proposer un STRE dans une perspective de haute disponibilité :

- indépendance de l'exécution des tâches de criticités différentes en introduisant la notion de **partition**. Une partition isole les données et programmes d'une application du point de vue mémoire mais aussi du point de vue temporel : chaque partition dispose de son propre espace d'adressage et de créneaux de temps. Elle ne peut empiéter ni sur les créneaux temporels ni sur l'espace mémoire des autres partitions.

- ordonnancement cyclique : une partition peut être activée une ou plusieurs fois par cycle. A l'intérieur d'une partition différentes tâches peuvent être ordonnancées de façon périodique ou non, la norme attend un ordonnancement par priorité et préemptif.

- La communication entre partitions se fait par messages, les messages sont reçus et émis par les applications sur des ports, le support de communication est géré par le système, les applications ne voient que les ports.

- Les services fournis par le système sont aussi isolés dans une partition spécifique. Traitement des pannes par l'implantation du *Health Monitor*.

JVM version RTSJ

- RTSJ, *Real-Time Specification for Java* est une spécification de fonctionnalités temps réel pour les JVM (http://www.rtj.org) :
  - Les apports de cette spécification sont orientés vers le déterminisme temporel :
    - L'ordonnancement doit être préemptif, avec une gestion FIFO des threads ayant la même priorité, la JVM ne peut changer la priorité d'un thread que dans le cas de l'inversion de priorité,
    - Gestion des threads au niveau utilisateur :
      - caractérisation fine des threads temps réel en précisant leur coût, leur échéance et leur période,
      - création de threads de priorité supérieure à celle du *garbage collector*.
      - service de contrôle d'admission : on peut demander le calcul de l'ordonnançabilité d'un jeu de threads.
- RTSJ propose aussi des outils pour construire des ordonnancements RMS, EDF ou LLF, pour créer des zones de mémoire qui ne sont pas gérées par le *garbage collector*, et met à disposition des timers haute résolution déterministes ainsi que des méthodes pour la gestion des événements asynchrones.

Plan

1. Architectures embarquées : de l'application au matériel
    a. Les 3 niveaux : application, support d'exécution, matériel
    b. L'architecture logicielle : un système réactif
    c. L'architecture logicielle : adaptation des modèles
    d. Les domaines d'application
2. Les supports d'exécution
    a. Notions de base : priorité, préemption
    b. Evolution
    c. Les différents modèles
    d. Quelques points techniques
3. **Les supports d'exécution disponibles**
    a. Exemples
    b. Critères de choix

Comment choisir le support d'exécution

- Plus est grande la variabilité des différents éléments (nombre de tâches, WCET, gigues, etc), plus il faudra un support d'exécution complexe :
    - pas de variabilité, connaissance a priori de toutes les contraintes --> ordonnancement hors ligne et écriture d'une séquence de fonctions qui traitent les réactions aux évènements,
    - variabilité importante, comment choisir (si on peut choisir) le support d'exécution :
        - sur quelle cible (BSP) va être déployée l'application ?
        - taille mémoire nécessaire en RAM et en ROM (*footprint*),
        - pilotes de périphériques disponibles, facilité de leur écriture,
        - quels langages de programmation sont disponibles ?
        - outils de développement proposés,
        - accès ou non au code source, support technique, maturité,

Surcoûts temporels dûs au Support d'exécution

- Une fois choisi le support d'exécution, il faut en vérifier les caractéristiques, par exemple :
  - Changement de contexte : **ne se fait jamais en temps nul**, il faut le prendre en compte dans le WCET,
  - Partage des ressources : il faut connaître la borne temporelle des opérations P et V sur les sémpahores dans tous les cas (standard, PCP, PIP),
  - Gestion de la hiérarchie de mémoire : attention à la pagination (mais en général inhibée en embarqué), fonctionnement des mémoires caches,
  - Entrées-sorties à bas niveau : attention aux priorités, échéances,

Quelques critères pour la conception(1)

- Identifier le type des évènements externes et celui des réponses à leur apporter :
    - type d'événement : périodique, alarme, requête sporadique, combien d'échéances peut-on manquer ?
    - type de réponse : temps de réponse borné, commande périodique,…
- Le modèle applicatif déduit est-il proche de celui fournit par le support d'exécution :
    - si non : adapter l'implémentation du modèle applicatif au modèle proposé par le support d'exécution,
    - si nécessaire : adapter également cette implémentation au type du matériel (event/time triggered),
- Choix des algorithmes :
    - ils doivent se satisfaire des performances (vitesse, taille) des ressources offertes par la plate-forme matérielle (mémoire, …),

Quelques critères pour la conception(2)

- Gestion mémoire :
    - pour le déterminisme temporel des accès à la mémoire:
        - pré-allocation statique des ressources (données, threads), pas d'allocation dynamique (new, malloc)
        - temps d'allocation proportionnel à l'espace d'adressage mémoire demandé (cf. RTSJ),
        - pas d'édition de liens dynamique, mais édition de liens statique,
- Gestion des E/S :
    - elles sont très couteuses, donc il faut les limiter. La fiabilité évitera les signalisations d'erreurs,

Plan

1. Architectures embarquées : de l'application au matériel
    a. Les 3 niveaux : application, support d'exécution, matériel
    b. L'architecture logicielle : un système réactif
    c. L'architecture logicielle : adaptation des modèles
    d. Les domaines d'application
2. Les supports d'exécution
    a. Notions de base : priorité, préemption
    b. Evolution
    c. Les différents modèles
    d. Quelques points techniques
3. **Les supports d'exécution disponibles**
    a. Exemples
    b. Critères de choix
4. **Perspectives**

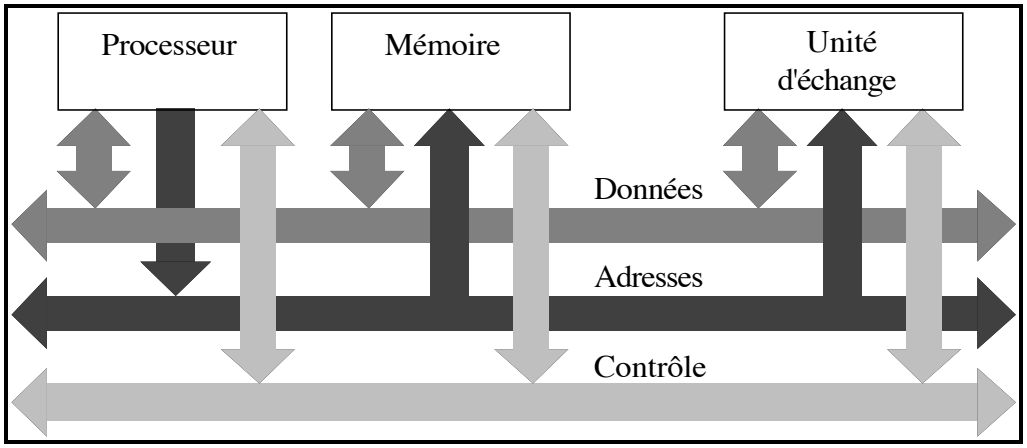
Perspectives : support d'exécution

- Du plus petit au plus grand :
    - Supports miniatures pour les réseaux de capteurs : TinyOS de Berkeley,
    - Systèmes embarqués distribués à grande échelle : RT CORBA,

- Ordonnancement : notions de partitions (ARINC 653), ordonnancement dynamique et adaptable (*flexible scheduling, power aware scheduling*),
- Gestion mémoire déterministe (cf. les outils RTSJ),
- Gestion de l'énergie,

Perspectives : matériel

- Modèle très simplifié des ressources matérielles :



- Chaque composant doit avoir caractéristiques temporelles déterministes :
  - bus CAN, bus TTP(Time Triggered Protocol), protocole TDMA,…
  - gestion mémoire déterministe (cf. caches),
- Compromis matériel/logiciel :
  - seules les fonctions simples et pérennes sont réalisées en matériel,
  - tendance aux processeurs généralistes et aux réalisations logicielles,
  - le matériel assure la performance, le logiciel apporte portabilité, flexibilité