Systèmes Temps Réels

Claude BARON

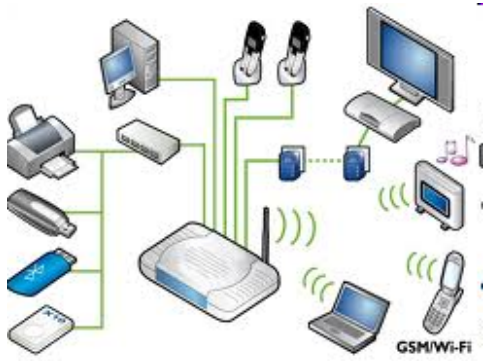
INSA – LAAS/CNRS

claude.baron@insa-toulouse.fr

Préambule

Du métro sans conducteur

au pilote automatique dans les avions



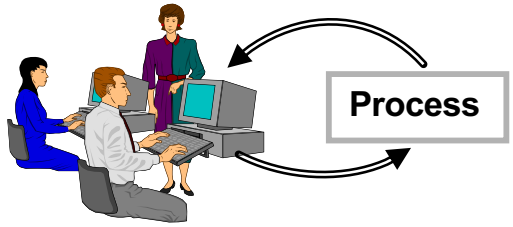
les systèmes temps réel font partie de notre vie quotidienne

- Applications dans divers secteurs :
  - contrôle industriel,
  - transport,
  - robotique
  - télécommunications,
  - systèmes de sécurité,
  - domotique...

Diversification des systèmes temps réel

La taille et la nature des systèmes temps-réel sont très divers...

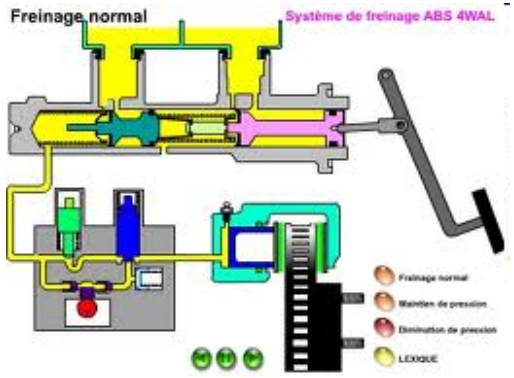
- Les plus grands (point de vue géographique) : systèmes de contrôle télémétrique (secteurs électricité, pétrole, transport ferroviaire) – commande et contrôle de sites éloignés à partir d'une unique salle de supervision



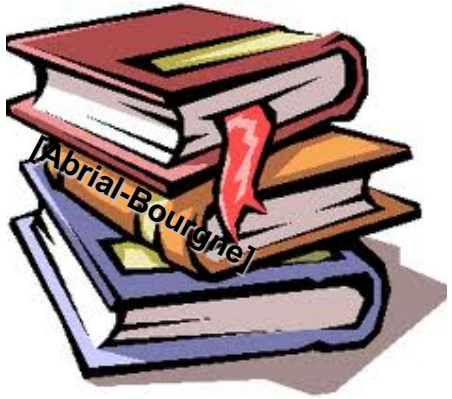
- Plus petits (en taille) mais plus complexes (nature des lois de commande), les systèmes de contrôle de missile ou de navigation aérienne se différencient des précédents :
  - ordre de grandeur des temps de réponse (milliseconde ⇒ radar, heure ⇒ contrôle de réactions chimiques)
  - criticité des fonctions qu'ils remplissent dans leur environnement

Préambule

- Concept passé dans le langage courant, parfois galvaudé
- Opacité de sens d'autant plus surprenante que nous utilisons de plus en plus de moyens pilotés par des systèmes temps réel :
  - ABS de voiture, robotique, gestion des portes de métro ou de train d'atterrissage des avions
  - les chaînes de fabrication agroalimentaire
  - …

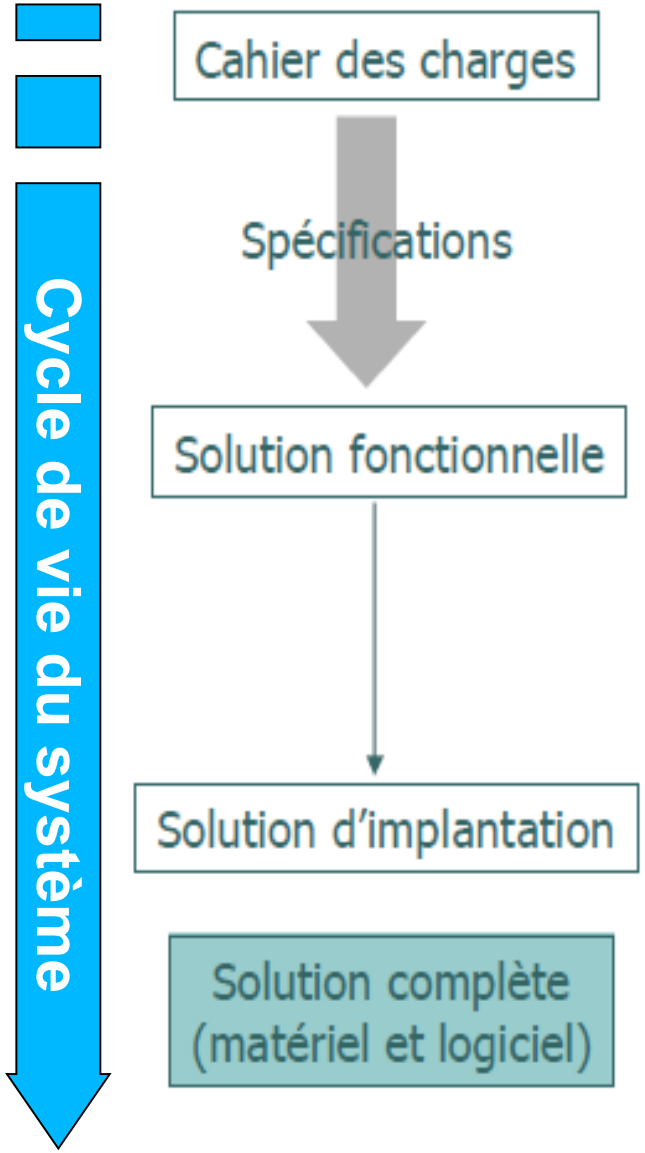


Sur la base de cette définition….



Un système fonctionne en temps réel s ’il est capable d ’absorber toutes les informations d ’entrée sans qu’elles ne soient trop vieilles pour l ’intérêt qu’elles présentent, et par ailleurs de réagir à celles-ci suffisamment vite pour que cette réaction ait un sens.

Progression du cours, TD, TP



- **Introduction**
  - Système temps réel /embarqué : définitions, exemples
  - Le temps réel : définitions, caractéristiques importantes
  - La criticité des systèmes

Cours

- **Spécification d'un système temps réel**
  - Définition du système, du contexte opérationnel
  - Définition des exigences
  - Modélisation du comportement attendu
- **Evaluation, vérification a priori du comportement**
  - Ordonnancement
  - Simulation des modèles

TD

- **Conception d'un système temps réel**
  - Modélisation et évaluation d'architectures
  - Choix d'implantation : OS, langage, support d'exécution, …
  - Exécutifs temps réel : définitions, architectures, caractéristiques
  - Implantation sur un OS temps réel : principes, intérêts, exemples
- **Implantation : programmation sur un OS temps réel**
  - Xenomai : gestion des tâches, communication/synchronisation, interruptions,
    gestion du temps

TP

- **Vérification a posteriori** : simulation, mesures, tests, ….

Un Système Temps Réel, c'est quoi ?

- Un STR se compose d'un système ou d'une application devant répondre en un temps fini et spécifié à des stimuli générés par le monde extérieur.
- Exemples :
  - contrôleur de vitesse pour voiture,
  - système de surveillance d'une centrale nucléaire
  - contrôleur de machine a laver…..

→ les STR se caractérisent par les conséquences dramatiques qui peuvent survenir lorsque les résultats obtenus ne satisfont pas certains critères de **cohérence logique** mais surtout **temporelle**.

L'exactitude en terme de comportement du système ne dépend pas seulement des **résultats logiques** de calculs informatiques mais aussi des **instant physiques** auxquels ces résultats sont produits

Définition d'un système temps réel

Il existe une multitude de définitions.

Pouget (vision ordonnancement) :

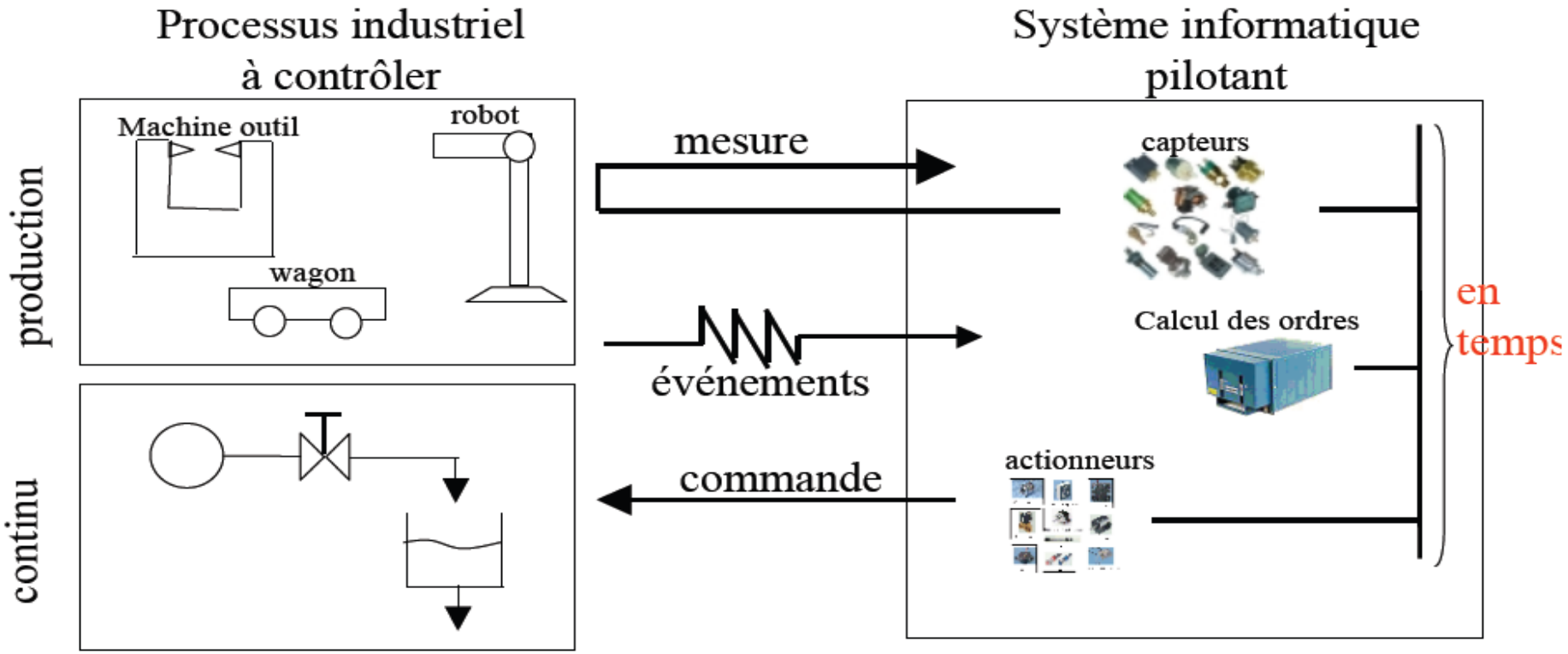
> « Le temps réel est le pilotage à tout instant d'un système évolutif dans le temps »

Elloy (vision concurrence et communication, automobile) :

> « Peut être qualifiée de temps réel (ou "temps contraint", ou "réactif") toute application dont le déroulement est assujetti à l'évolution dynamique d'un procédé extérieur qui lui est connecté et dont il doit contrôler le comportement ».

Définition d'un système temps réel

Est appelé *temps réel* le comportement d'un système informatique dont le fonctionnement est assujetti à l'évolution dynamique d'un procédé industriel connecté à lui. On dit que le processus est contrôlé, piloté ou supervisé par le système qui *réagit* aux changements d'état du processus.



In computer science, **real-time computing**, or **reactive computing**, is the study of hardware and software systems that are subject to a "real-time constraint"— i.e., operational deadlines from event to system response.

Conséquences de la définition

- Un STR reçoit des événements émanant du procédé à contrôler ; ces événements peuvent être périodiques ou non
  - ⇒ le système doit réagir avant un délai ou une date fixée
  - ⇒ aucun événement ne doit être raté par le système
- L'important est de respecter l'échéance
  - ⇒ Ne pas réagir à temps (résultat juste hors délai) peut être considéré comme une défaillance catastrophique
  - ⇒ Un système Temps Réel est souvent critique et doit souvent être tolérant aux pannes

Différents types de systèmes

La dynamique de réponse comme critère de distinction :

- les systèmes *transformationnels* : le temps de réponse importe peu
- les systèmes *interactifs* (ou en-ligne) : on souhaite une réponse dans un délai court, le rythme de l'interaction est déterminé par le système
- les systèmes *temps-réel* : toute réponse doit être obtenue dans un laps de temps court et prédéfini, notion de promptitude, en interaction permanente avec l'environnement, **le rythme de l'interaction est déterminé par l'environnement**

=> capacité à pouvoir appréhender un flux d'évènements asynchrones issus d'un processus, sans perdre un seul de ces évènements et de traiter chacun d'eux en un temps déterminé.

2 concepts majeurs associés à la définition

« JUSTE ET À TEMPS »

- Axiome :

  ‘L’ensemble des traitements à réaliser pour générer les sorties est fait correctement dans le temps imparti, à partir de données d’entrée valides’

- => respecter les contraintes temporelles de l’application
  => travailler avec des données cohérentes
    - spatialement
      - toute image de la donnée doit être la même dans tout le système
    - temporellement
      - toute évolution d’une donnée doit se propager dans le système dans un temps borné connu à l’avance, de préférence le plus faible possible

  => respecter la durée de vie de la donnée

« MÊME CAUSE MÊME EFFET »

- L’axiome précédent se répète toujours, quelle que soit l’occurrence des événements dans le système

Plusieurs notions de temps réel

- **‘Hard’** Real Time : garantit que les tâches critiques finissent à temps

- **‘Soft’** Real Time : moins restrictif; une tâche critique est prioritaire sur les autres et conserve cette priorité jusqu'à ce qu'elle se termine

→ lié à la notion de ‘criticité’ du système

Real-time systems, as well as their deadlines, are classified by the consequence of missing a deadline.

The goal of a **hard real-time system** is to ensure that all deadlines are met For **soft real-time systems** the goal becomes meeting a certain subset of deadlines in order to optimize some application specific criteria.

Notion de criticité

La dépendance aux contraintes temporelles peut parfois prendre un caractère critique

Criticité

gravité de la situation qui résulterait d'un manquement au respect des spécifications temporelles

Le degré de criticité est fonction des conséquences des déviations par rapport à un comportement nominal

Ces conséquences peuvent concerner la sûreté des personnes et des biens, la sécurité, l'accomplissement des missions, la rentabilité économique.

Mission critical refers to any factor of a system (equipment, process, procedure, software, etc.) whose failure will result in the failure of business operations.

Notion de criticité

Deux notions de criticité face aux manquements d'une échéance :

Contraintes temps réel **strictes** : le dépassement d'une échéance est catastrophique

Contrainte temps réel **relatives** : le dépassement d'une échéance peut être toléré (dans une certaine mesure)

=> **Temps réel Dur** / **Temps réel Lâche**

=> dans le cas des système Temps Réel Dur, on cherchera à obtenir un comportement **prévisible**, **déterministe** et **fiable**
=> utilisation de techniques mathématiques (ordonnancement, évaluation des pires cas…)

=> dans le cas des systèmes Temps Réel Lâche, on cherchera à minimiser la probabilité de rater une échéance plusieurs fois de suite...

Hard vs. Soft Real Time Systems

Temps Réel Dur versus Temps Réel Mou

Caractéristique	Temps Réel Dur	Temps Réel Mou
Temps de Réponse	Exigence stricte	Souhaitée
Peak-load performance*	Prédictible	Dégradée
Contrôle du rythme	Environnement	Ordinateur
Sûreté	Souvent critique	Non critique
Dimension des fichiers de données	Petite/moyenne	grande
Type de redondance	Active	Check-point recovery
Intégrité des données	Court terme	Long terme
Détection d'erreurs	Autonome	Assisté par l'utilisateur

* The maximum instantaneous load or the maximum average load over a designated interval of time

Evaluation du respect des délais

- Prouver théoriquement que les limites temporelles ne seront jamais dépassées quelle que soit la situation pour un système temps réel prétendu strict
- vérification appelée "**test d'acceptabilité**" ou "analyse de faisabilité"
- fait appel à la théorie de l'ordonnancement
- dépend de l'ordonnanceur utilisé et des caractéristiques des tâches du système :
  - ex : pour un système souple, on pourra se contenter de mesures statistiques obtenues sur un prototype

Evaluation du respect des délais

- **Condition de charge**
  - Pour tout système de *n* tâches, la condition suivante est nécessaire mais pas suffisante à sa faisabilité :

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

Avec *Ci* le temps de calcul de la tâche n° *i* et *Ti* sa période

Une valeur supérieure à 1 signifierait que le système nécessite plus de temps d'exécution que le processeur ne peut en fournir

- **Temps de réponse dans le cas le plus défavorable (worst case)**
  - la « plus longue durée entre l'activation de cette tâche et son instant de terminaison » parmi tous les scénarios possibles d'exécution du système
  - Une tâche est faisable si son temps de réponse dans le pire des cas est inférieur ou égal à son échéance
  - Un système est faisable si toutes les tâches qui le composent sont faisables

Le temps réel et l'embarqué

Système embarqué :

- dispositif informatique intégré au dispositif physique dont il assure le contrôle et (ou) la commande et dont il partage les contraintes d'environnement (spatial, automobile...)

➲caractérisé par

- l'environnement dans lequel il fonctionne (température, humidité, radiations, vibrations, chocs…,⇒ taille, poids…)
- la performance qu'on attend de lui
- ses interfaces avec l'environnement

Embarqué : impératifs spécifiques

- **criticité** : les systèmes embarqués sont souvent critiques, et les systèmes critiques sont presque toujours embarqués
- **réactivité** :
  - interagir avec l'environnement à une vitesse imposée par celui-ci
  - induit des impératifs de temps de réponse => l'informatique embarquée est souvent basée sur un système temps réel
- **autonomie** : remplir la mission pendant de longues périodes sans intervention humaine
  - intervention humaine impossible
  - réaction humaine trop lente ou insuffisamment fiable
- **robustesse, sécurité et fiabilité** : environnement souvent hostile, pour des raisons physiques (chocs, variations de température, impact d'ions lourds, ...) ou humaines (malveillance)

  => la sécurité (résistance aux malveillances) et la fiabilité (continuité de service) sont souvent rattachées à la problématique des SE
- **contraintes non fonctionnelles** (occupation mémoire, consommation d'énergie, ...)

Exécutifs Temps Réels

1. Introduction : pourquoi un RTOS ?

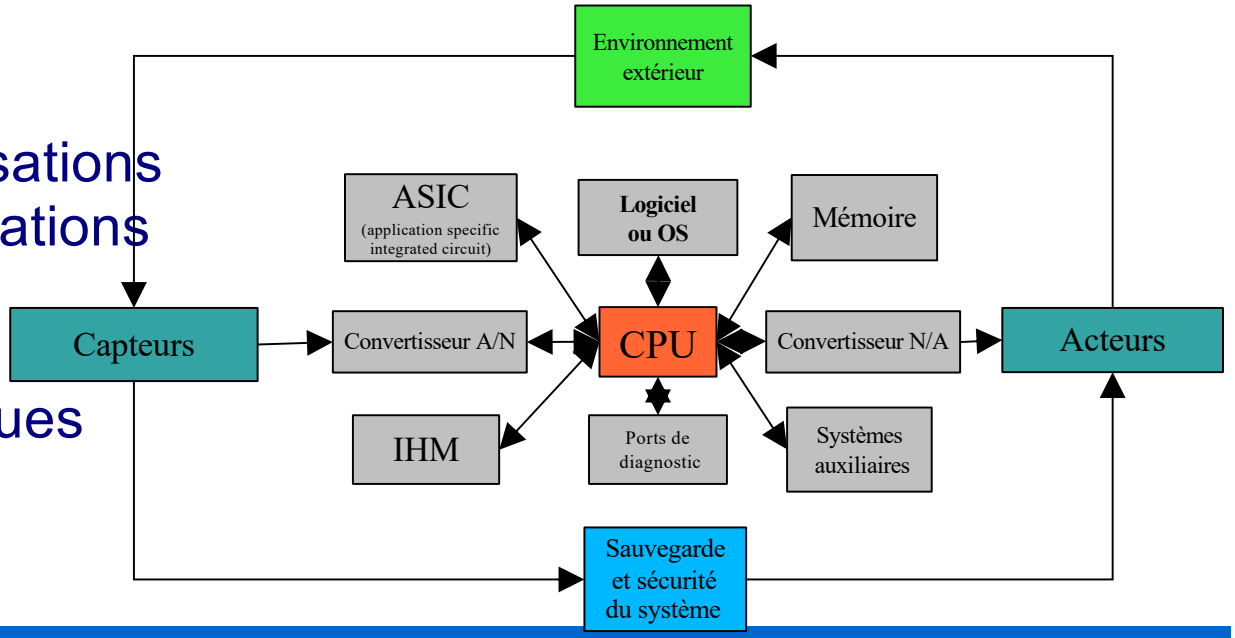
Le système informatique temps réel est constitué d'une association logiciel/matériel

Rôle du logiciel : (entre autres) gestion adéquate des ressources matérielles en vue de remplir certaines tâches dans des limites temporelles bien précises

=> RTOS : offre des services au(x) logiciel(s) d'application services basés sur les ressources disponibles au niveau du matériel

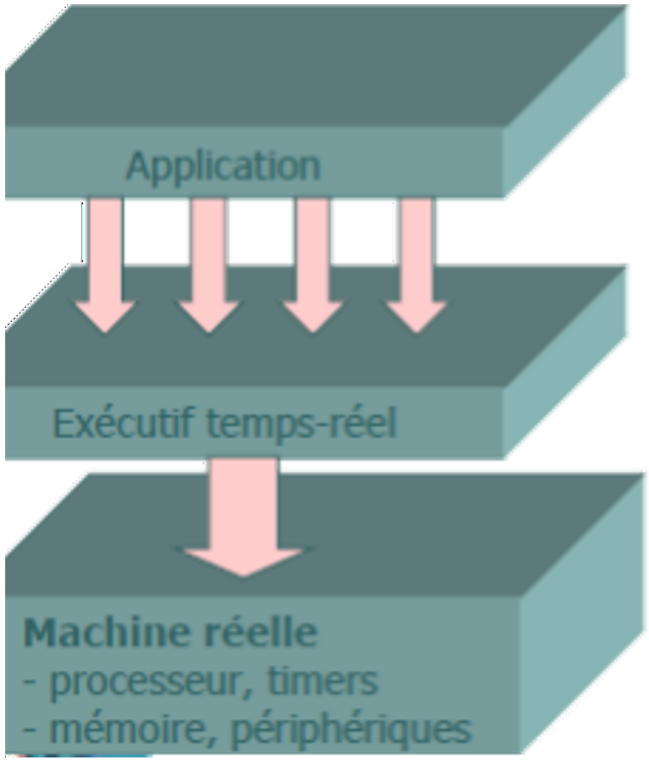
Exemples :

- Gestion des tâches
- Gestion des synchronisations
- Gestion des communications
- Gestion de la mémoire
- Gestion du temps
- Gestion des périphériques



1. Introduction : pourquoi un RTOS ?

Autre intérêt : portabilité, réutilisation des applications



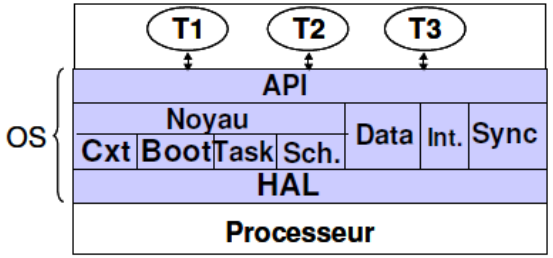
Appels systèmes (primitives)

Ex : gestion de tâches (création, arrêt)
Ex : gestion du temps (attente d'un délai)

Gestion du matériel

Ex : sauvegarde de registres pour préemption
Ex : écriture registre timer pour délai

⇒ Plus besoin de manipulations du matériel



Source: Thèse de doctorat de Gabriela Nicolescu, Laboratoire TIMA, France, 2002

1. Introduction : généralités RTOS

Un RTOS est un OS dont les caractéristiques doivent lui permettre de supporter un Système TR

- **RTOS = programme dont le rôle est**
  - d'ordonnancer l'exécution des tâches
  - de gérer les ressources du système
  - de fournir les éléments de base pour développer les applications
- **Organisé autour :**
  - d'un noyau fournissant les algorithmes minimum pour l'ordonnancement et la gestion des ressources
  - éventuellement de services (gestion des systèmes de fichier, accès réseau, tout autre service requis par l'application)

1. Introduction : généralités RTOS

Pour garantir le respect de contraintes temporelles, il est nécessaire que :

- les différents services et algorithmes utilisés s'exécutent en temps borné
  - => le RTOS doit être déterministe et préemptif pour donner la main durant le prochain tick à la tâche de plus forte priorité prête
  - on peut en théorie dans ce cas prédire l'évolution du STR
- les différents enchaînements possibles des traitements garantissent que chacun de ceux-ci ne dépassent pas leurs limites temporelles

=> Caractéristique importante d'un RTOS = temps de réponse prévisible à un stimulus externe.

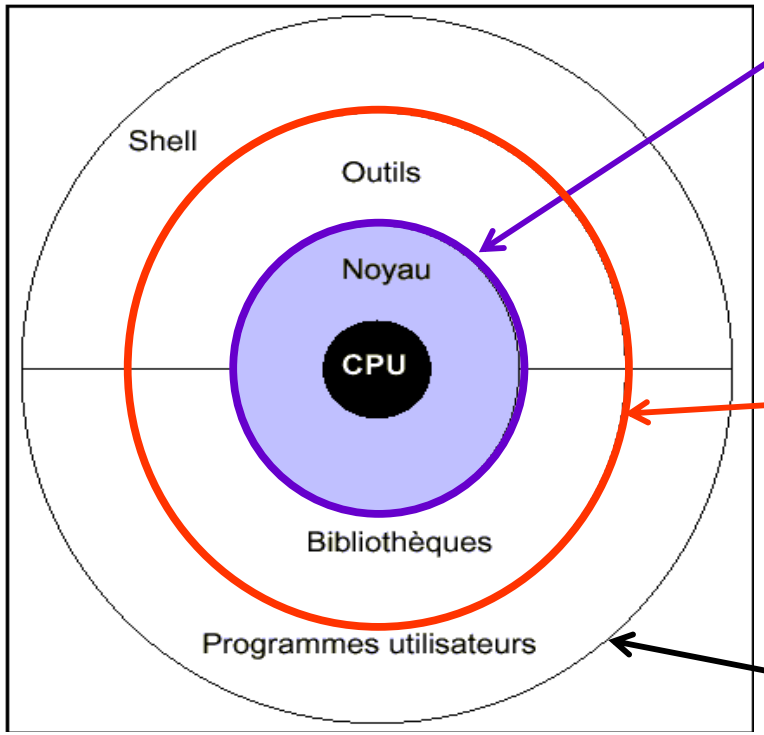
- Si un périphérique génère une interruption, le RTOS doit répondre et démarrer le service à l'intérieur d'une période de temps connue, et ce peu importe la charge du processeur

un OS est un RTOS lorsque le changement de contexte et le temps de réponse à une interruption sont garantis à l'intérieur d'une période de 10 µs

2. Définitions : structure d'un RTOS

Organisé autour :

d'un noyau : fournit les algorithmes minimum pour
l'ordonnancement et la gestion des ressources
de services
(gestion systèmes de fichier, accès réseau, …)



Le **noyau** gère
- ✓ les périphériques
- ✓ l'exécution des programmes (processus)
- ✓ la mémoire attribuée à chaque processus
- ✓ l'ordonnancement des processus
- ✓ la synchronisation et communication entre processus
- ✓ la gestion des protocoles réseau
- ✓ logiciel compact, de petite taille (moins de 10 Ko)

⇒minimum logiciel pour pouvoir faire du temps réel :
- ordonnanceur
- gestion de tâches
- communications inter-tâches
- allocation des ressources, des interfaces matériel, de la sécurité

⇒ système plutôt limité mais performant.

L'**exécutif**
- ✓ construit sur le noyau
- ✓ contient des fonctionnalités comme
  - gestion des entrées/sorties
  - mécanismes de coopération évolués (construits à partir de ceux du noyau)
  - gestion d'un système de fichiers
  - facilités (eg connexion distante).

Le système d'exploitation
- inclue tous les outils de développement et de mise au point de l'application (compilateur, metteur au point, simulateur, analyseur...)

2. Définitions : rôle d'un exécutif TR

Un exécutif temps réel peut être

event driven : system switches tasks when an event of higher priority needs service

time sharing : system switches tasks on regular clock interrupts and on events

=> Les appels à l'exécutif sont la conséquence :

des occurrences d'événements issus du procédé

l'appel à l'exécutif est effectué par la procédure de réception et de traitement de l'interruption matérielle associée à ces événements

du temps

l'appel à l'exécutif est provoqué par l'interruption régulièrement engendrée par une horloge temps réel équipant le calculateur

des tâches elles-mêmes

2. Définitions : rôle d'un exécutif TR

Rôles

- ordonnancer les exécutions des tâches
- protéger l'accès aux ressources partagées
- rôle centralisateur : interface qui
  - aiguille les événements reçus du procédé vers les tâches qui les attendent
  - déclenche le réveil des tâches en attente d'un délai ou d'une heure de démarrage
  - reçoit et retransmet des signaux de synchronisation ou des données entre des tâches asynchrones

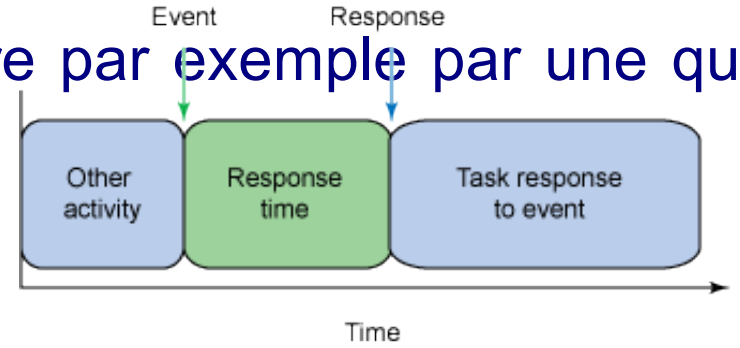
Offre des services accessibles par l'utilisateur dans ses tâches, de différentes natures

- gestion des tâches : activation, suspension, reprise, terminaison forcée …
- gestion des événements matériels (interruptions)
- Synchronisation : émission/réception par les tâches de signaux « internes »
  => L'exécutif offre des services de génération et d'attente de ces « événements logiciels »

2. Définitions : caractérisation RTOS

Illustration de la différence entre un RTOS et un OS standard

- Vous êtes dans un jeu video dans lequel chacune de vos actions déclenche l'exécution d'un programme
- Si le jeu est géré par un RTOS vos actions vont être exécutées avec un léger retard, connu et borné
- Si le jeu est géré par un OS standard, ce retard n'est pas fiable, les effets des actions risquent d'être désynchronisés
- Pour garantir le temps de réponse, les programmes TR et leurs OS doivent gérer en priorité l'actualisation des échéances des actions
- Cela va se traduire par exemple par une qualité d'image moindre éventuellement



Interrupt latency and response time

Response time should be deterministic and operate within a known worst-case time

3. Services RTOS

Services fournis par un RTOS (vue détaillée)

- Gestion des tâches
- Relations inter-tâches :
  - Gestion des synchronisations
  - Gestion des communications
- Gestion des interruptions
- Gestion de la mémoire
- Gestion du temps
- Gestion des périphériques

3. Services RTOS : gestion des tâches

Terminologie

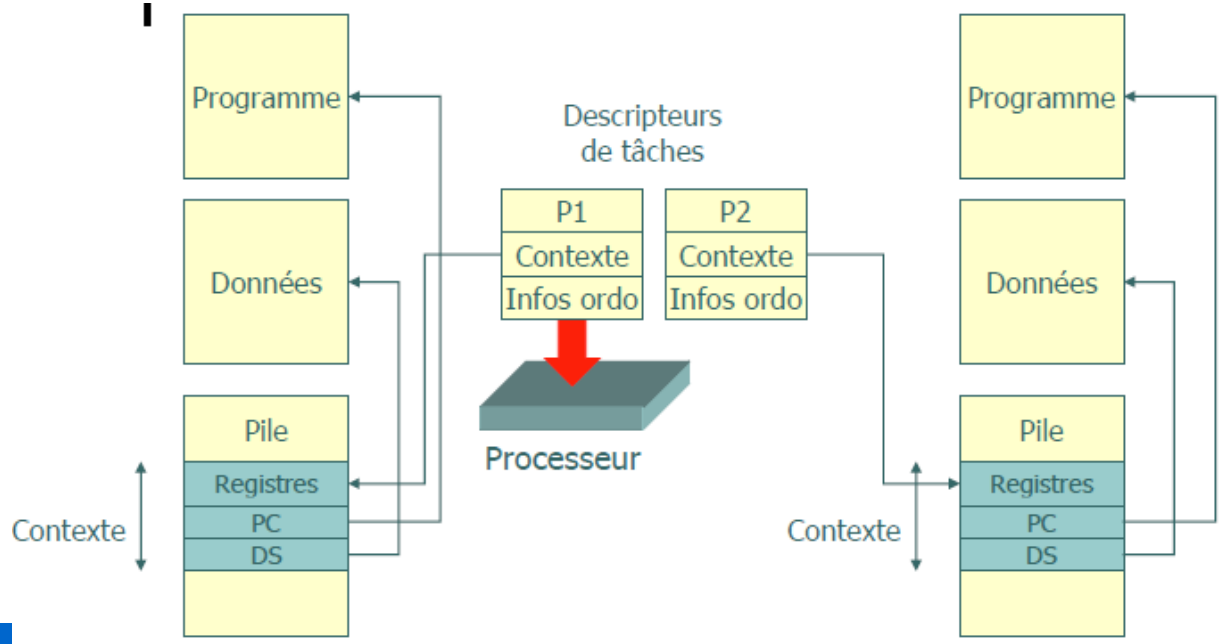
Programme : Entité constituée des instructions à dérouler pour effectuer un calcul (statique)

Tâche : Entité dynamique composée de

Programme (composante statique)

Données, pile d'exécution, contexte (composante dynamique)

Descripteur : structure de donnée regroupant les informations sur une tâche à un instant donné (nom, contexte d'exécution, …)



3. Services RTOS : gestion des tâches

Structure de l'exécutif

L'ordonnanceur

le cœur du noyau

agit sur des entités ordonnançables (tâches)

- threads
- processus

gère plusieurs tâches à la fois (multitâche)

- doit donner la CPU à la bonne tâche au bon moment (ordonnancement)

met en œuvre les changements de contexte si nécessaire

- sauvegarde et restauration du bloc de contrôle de tâche (TCB)

Le répartiteur (dispatcher)

partie de l'ordonnanceur qui exécute les changements de contexte et modifie le cours de l'exécution

le cours de l'exécution peut être consacré à :

- tâche utilisateur
- ISR
- noyau

passage du cours de l'exécution au noyau dès qu'une tâche ou un ISR fait un appel système

ensuite une autre (même ?) tâche reprend la main

un ISR court-circuite le répartiteur jusqu'à la fin de son exécution

3. Services RTOS : gestion des tâches

Ordonnancement

Types d'ordonnanceurs

Ordonnanceurs non préemptifs (séquenceurs)

- Jouent une séquence d'exécution générée statiquement (non préemptif)
- Noyaux propriétaires essentiellement (non commerciaux), hormis OsekTime (automobile)

Ordonnanceurs préemptifs

- Ordonnanceurs à priorités
  - Priorité par tâche, allocation du processeur à la tâche la plus prioritaire, préemptif
  - Quasi-totalité des exécutifs temps-réel
- De type temps-partagé
  - Allocation du processeur par tranche de temps (préemptif)

4. RTOS basics facilities : gestion des tâches

- Ordonnancement
  - Les exécutions des tâches doivent respecter des contraintes temporelles dictées par les dynamiques du procédé qu'elles pilotent ou contrôlent
  - Ordonnancer =
    1. déterminer la politique d'allocation des tâches sur les supports d'exécution (processeurs, contrôleurs) pour garantir ces contraintes temporelles
    2. puis implémenter cette politique : déterminer le mécanisme à intégrer dans l'exécutif qui lui permettra de décider de l'ordre des tâches lors de l'évolution de l'application (observation des événements émis par le procédé)
  - Exigence du respect des contraintes pas systématique pour toutes les tâches
    - tâches **critiques** : doivent impérativement être exécutées dans des fenêtres temporelles conditionnées par les dynamiques du procédé
      - le non-respect des contraintes des tâches critiques peut compromettre la stabilité du procédé, sa sécurité ou celle de son environnement
      - celui des tâches non critiques peut altérer certaines prestations du procédé ou en dégrader les performances

3. Services RTOS : gestion des tâches

- Ordonnancement
  - une politique valide doit garantir ces contraintes pour toutes les combinaisons possibles d'événements
  - on peut tenter de garantir ce respect des contraintes de deux façons différentes
    - déterminer hors-ligne une séquence spécifique des exécutions des tâches qui soit valide pour l'application concernée
      - peut exiger l'énumération de toutes les combinaisons possibles d'événements et la recherche associée d'un ordre d'exécution des tâches qui, pour chacune de ces circonstances, garantisse le respect des contraintes de l'application
    - élaborer des algorithmes d'ordonnancement dont on puisse prouver qu'ils sont optimaux pour certains modèles d'applications et de contraintes, puis faire rejoindre l'application traitée à l'un de ces modèles
    - $\Rightarrow$deux façons différentes d'aborder le problème de l'ordonnancement
    - $\Rightarrow$peuvent produire des séquences d'exécution de tâches non comparables, bien que valides, et leurs mises en œuvre peuvent être très différentes

3. Services RTOS : gestion des tâches

Ordonnancement

à l'origine des deux grandes classes d'ordonnancement

ordonnancement « statique »

- effectue le calcul de la séquence d'allocation des processeurs à partir de la connaissance complète de toutes les caractéristiques des tâches, avant le lancement de l'application
- hypothèse réaliste pour de nombreuses applications « embarquées »
  - nombre « raisonnable » de tâches aux échéances différentes (moins de 100)
  - taux d'occupation des processeurs élevé
  - aucun dépassement d'échéance autorisé

ordonnancement « dynamique »

- alloue les processeurs à partir de la seule connaissance des caractéristiques des tâches en cours d'exécution et de celles en demande d'exécution au moment du choix
- fixe dynamiquement son choix en ignorant l'existence des tâches futures
- la séquence construite varie dans le temps en fonction des occurrences d'événements, donc des réveils et arrêts des tâches de l'application

3. Services RTOS : gestion des tâches

Niveaux de parallélisme

- Parallélisme réel : Systèmes mutiprocesseurs uniquement
- Pseudo-parallélisme : le processeur exécute successivement plusieurs traitements.
  - Les traitements sont exécutés en séquence : ordonnancement non préemptif
  - Un traitement peut être interrompu par un autre traitement : ordonnancement préemptif

- Système multitâche monoprocesseur non préemptif

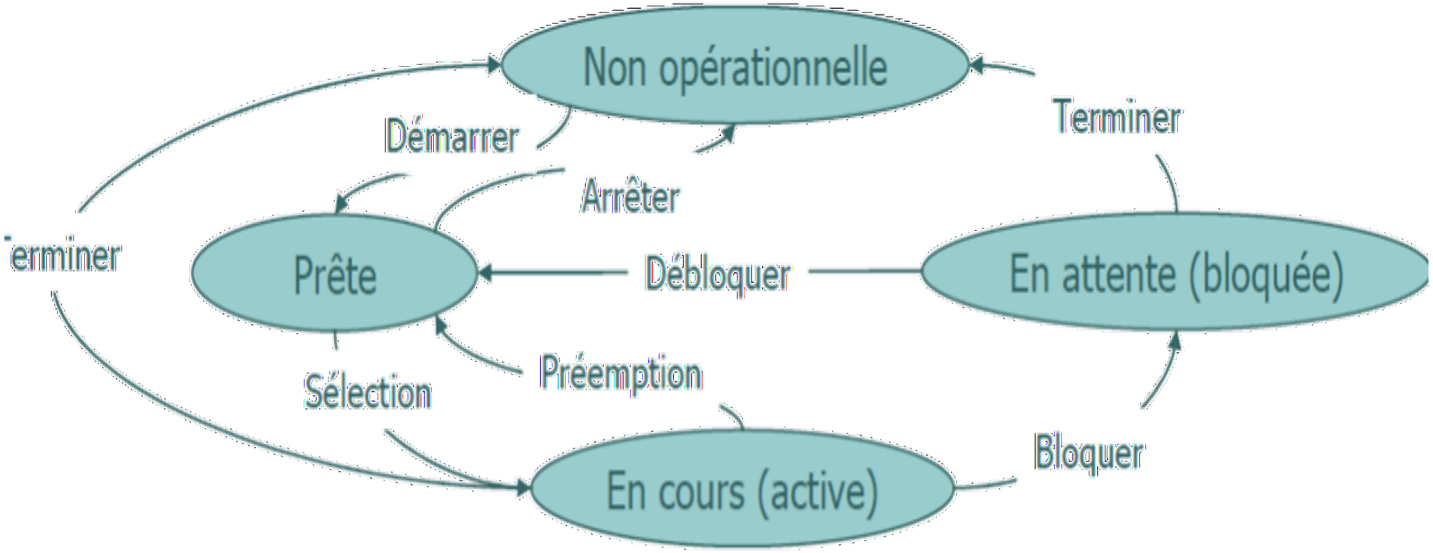
- Système multitâche monoprocesseur préemptif

- Système multitâche multiprocesseurs

3. Services RTOS : gestion des tâches

États d'une tâche

- Partage du processeur par plusieurs tâches
  - Notions de tâche prête et tâche active
- Primitives de synchronisation : blocage des tâches
- Diagramme de transition entre états



Terminer, Arrêter, Démarrer : primitives (⇒ appelées par les tâches) de gestion de tâches
Bloquer, Débloquer : primitives de synchronisation
Sélection, Préemption : décisions internes de l'ordonnanceur

3. Services RTOS : relations inter-tâches

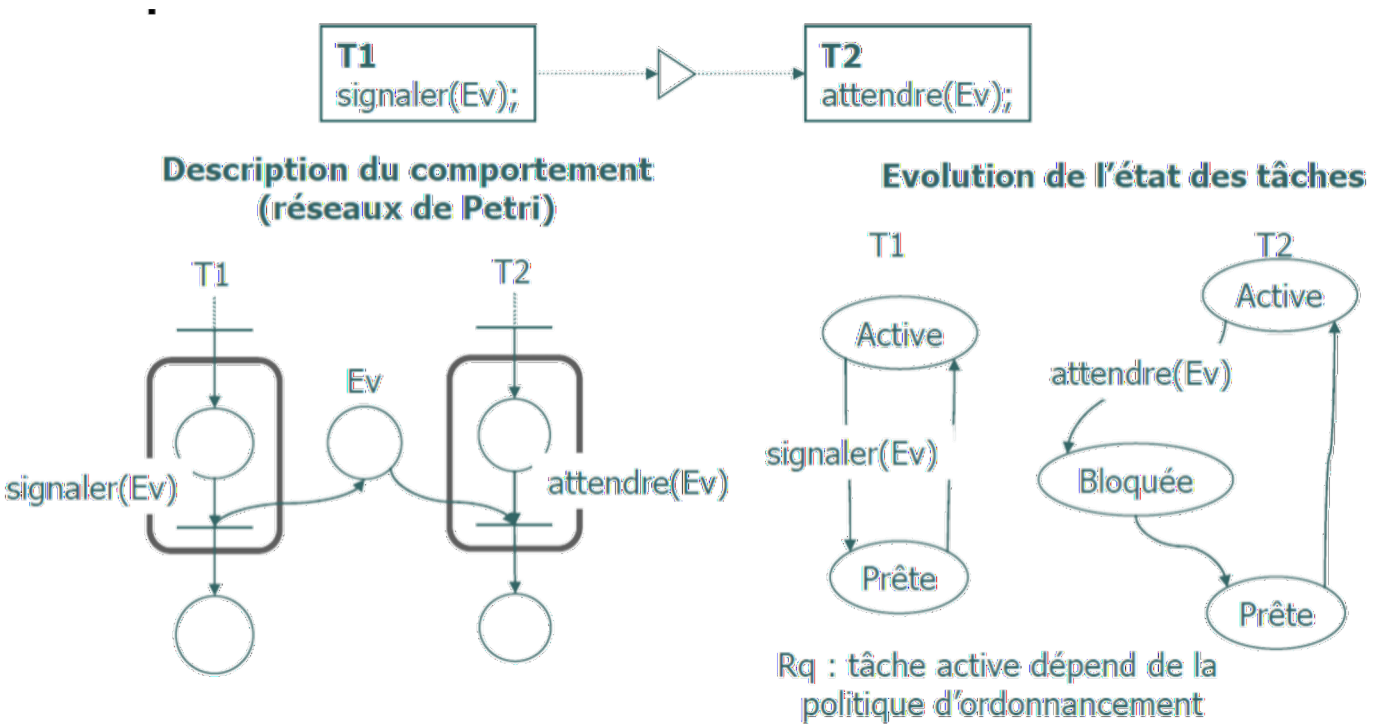
Relations inter-tâches de deux types
- Précédence (synchronisation par événement)
  - Notion d'ordre entre les traitements
- Partage de variables/ressources
  - Échanges d'informations ou de résultats

Primitives de synchronisation/communication offertes par les exécutifs temps-réel

3. Services RTOS : relations inter-tâches

Synchronisation

- Rôle : pour attendre un événement avant d'exécuter un calcul
- Origine des événements
  - Matériel : capteurs
  - Logiciel : synchronisation interne à une application



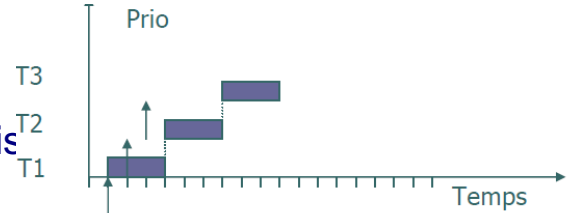
3. Services RTOS : relations inter-tâches

Synchronisation

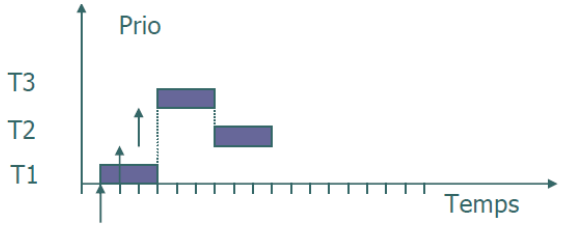
Synchronisation par sémaphores

Impact de la gestion des files d'attente

- Code des tâches
  - T1 : P(s); A1; V(s);
  - T2 : P(s); A2; V(s);
  - T3 : P(s); A3; V(s);
  - Arrivée de T1 en 1, de T2 en 2, de T3 en 3. prio(T1) < prio(T2) < prio(T3); Durée des Ai de 3;
- Cas d'une file gérée en FIFO



- Cas d'une file gérée par priorité crois



- File FIFO : équité, pas de famine
- File par priorité : moins équitable, mais favorise les tâ é aux systèmes temps-réel

=> Applicable à tout objet de synchro bloquante

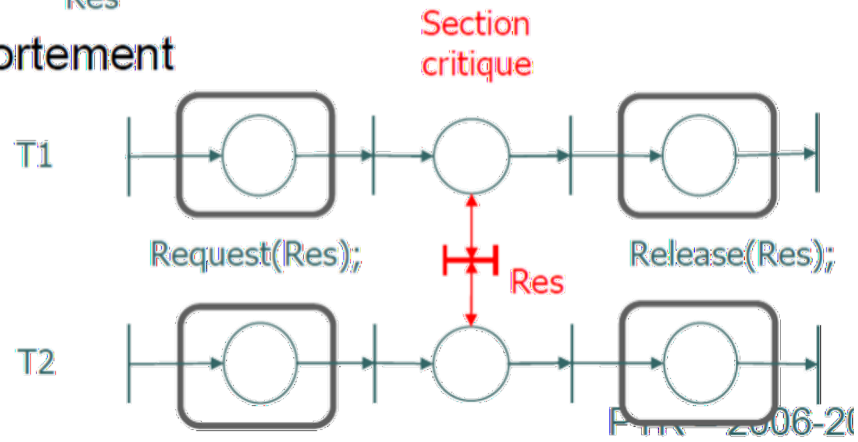
3. Services RTOS : relations inter-tâches

Partage de ressources

- Variable partagée → deux tâches ne doivent pas y accéder en lecture et écriture simultanément
- Notion de section critique, accès en exclusion mutuelle

- Modèle fonctionnel

- Comportement



3. Services RTOS : relations inter-tâches

Partage de ressources

Implantation d'une section critique

Suppression / masquage des interruptions

- Faible coût à l'exécution (cli/sti)
- Latence (gigue) dans le traitement des interruptions, voire perte d'interruptions
- Non valable en multiprocesseurs

→ Utilisable uniquement sur des sections critiques courtes, de taille bornée

- Augmentation temporaire de la priorité pendant la section critique
  - Retard dans le traitement des tâches non concernées par la ressource

Implantation par attente active

- Pas vraiment une bonne idée car
  - Monopolisation du processeur
  - Situation de famine quand une tâche prioritaire attend une ressource possédée par une tâche moins prioritaire : ne fonctionne pas !

```
void Request (char *V)
{
        while (*V) ; /* Attente ressource */
        *V =  1;
}
```

```
void Release (char *V)
{
        *V =  0; /* Libération */
}
```

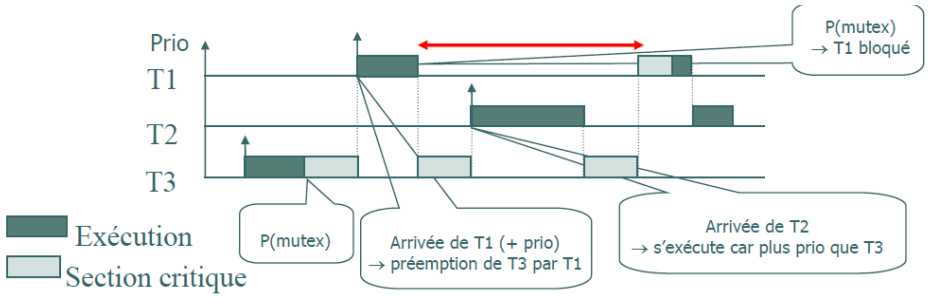
3. Services RTOS : relations inter-tâches

Partage de ressources

Implantation d'une section critique

Utilisation de sémaphores (attente passive)

- Principe :
  - Request(v) → P(s_v);
  - Release(v) → V(s_v);
- Sémaphores binaires
- Sémaphores à compteurs
- Sémaphores d'exclusion mutuelle (mutex) : sémaphores avec en plus les propriétés
  - Valeur initiale de 1 (pas de valeur en paramètre de la création)
  - Restriction des appels système pour que le Request (P) et le Release (V) soient faits par la même tâche (ex: VxWorks)
  - Possibilité d'emboiter dans la même tâche (sans blocage) des accès à la même ressource (ex: VxWorks)
- Problème d'inversion de priorité



T1 bloqué pendant toute la durée de T2. Durée blocage potentiellement infinie (multiples tâches de prio. interm).

7

4. Quelques standards de RTOS

POSIX

- POSIX (“Portable Operating Systems Interface”) is a standard for the function calls (API) of UNIX-like general purpose operating systems.
- Some specifications on real-time primitives too. Its definition of real time is quite loose: ability of the OS to provide a required level of service in a bounded response time.
- Many of the real-time POSIX extensions have already been implemented in RTLinux and RTAI

4. Quelques standards de RTOS

Linux for real-time and embedded
- general purpose operating system, with a non-pre-emptable kernel : poor RTOS
- Linux's basic user space scheduler is of the time slicing type: it gives more or less equal time slices to different tasks
- possible to change the priorities of user space tasks to some extent but not enough to make the scheduling deterministic.
- non-pre-emptable operations running in kernel space
- nobody can understand the kernel sufficiently well to be able to predict how long a certain operation is going to take

efforts towards the area of real-time and embedded systems
- Eliminating functionalities from the standard Linux kernel
- Patches
- Real-time patches underneath the Linux kernel : runs Linux as a low-priority process in a small real-time kernel.
- Ex. RTLinux (http://www.rtlinux.org/) and RTAI (http://www.rtai.org/)