« Systèmes Embarqués Temps Réels »

TELECOM Nancy
3éme année Approfondissement LE / Apprentissage

Vincent Bombardier
(MdC 61ème Section)

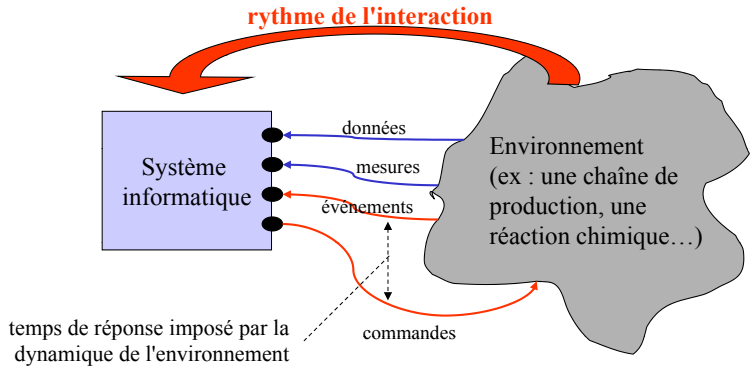
Centre de Recherche en Automatique de Nancy -UMR CNRS 7039-
Département: Ingénierie des Systèmes Eco-Technique
Projet Systèmes Intelligents Ambiants

ISET

Systèmes Temps Réels:
Plan du cours

Caractéristiques d'un STR
1.1. Définitions...

- Définition générale (CNRS 1988) :
  - Peut être qualifiée de "**temps-réel**" (ou "**temps contraint**", ou encore "**réactif**") toute application mettant en œuvre un système informatique dont le fonctionnement est assujetti à l'évolution dynamique de l'environnement qui lui est connecté et dont il doit contrôler le comportement.



Caractéristiques d'un STR
1.1. Définitions...

- Un peu de classification… ou un autre point de vue du temps réel.
  - Système Transformationnel :
    - activité de calcul, qui lit ses données et ses entrées lors de son démarrage, qui fournit ses sorties, puis meurt.
  - Système Interactif :
    - système en interaction quasi permanente avec son environnement, y compris après l'initialisation du système ; la réaction du système est déterminée par les événements reçus et par l'état courant (fonction des événements et des réactions passés) ; le rythme de l'interaction est déterminé par le système et non par l'environnement.
  - Système **Réactif** ou **Temps Réel** :
    - système en interaction **permanente** avec son environnement, y compris après l'initialisation du système ; la réaction du système est déterminée par les événements reçus et par l'état courant (fonction des événements et des réactions passées) ;
    - **mais le rythme de l'interaction est déterminé par l'environnement et non par le système.**

Caractéristiques d'un STR
1.1. Définitions...

- Un peu de classification… ou un autre point de vue du temps réel.

  Attention : les notions "Interactif" et Réactif" sont **relatives** à l'environnement considéré et à sa dynamique !

  => un système peut être considéré comme réactif face à un environnement à dynamique lente, mais seulement interactif face à un environnement à dynamique plus rapide.

  => **pas de notion absolue** ...

  => ... sauf à considérer un temps de réaction **nul** !

  => langages synchrones

Caractéristiques d'un STR
1.1. Définitions...

- Conséquences de la définition :
  - Un système Temps Réel 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

  ⇒ Autre formulation : *un système fonctionne en "Temps Réel" s'il est capable d'absorber toutes les informations d'entrée sans qu'elles soient trop vielles pour l'intérêt qu'elles représentent, et par ailleurs de réagir à temps pour que cette réaction ait un sens.*

  ⇒ Ne pas réagir à temps peut être considéré comme une défaillance catastrophique

  ⇒ Un système Temps Réel est souvent critique et doit souvent être tolérant aux pannes

  ⇒ mécanismes de tolérance aux défaillances (réplication, programmation N-version, vérification formelles…)

Caractéristiques d'un STR
1.1. Définitions...

- Relativité du temps réel:
  - Deux entités : l'environnement (ou le procédé) à contrôler et le système informatique Temps Réel
  - => deux temps : le temps de l'environnement et le temps du système Temps Réel
    - **Temps de l'environnement** = temps **chronométrique** (le temps réel)
    - **Temps du système informatique** = temps **chronologique**, constitué de la suite des événements ou des instructions du système (pas de temps réel vu par le système)
  - => Exigence du temps réel = concordance entre le temps chronométrique de l'environnement et le temps chronologique du système
  - => Le système informatique doit mettre ses actions en phase avec le temps chronométrique de l'environnement
    - où les actions du systèmes seront en fait ses "tâches" et ses "messages"
    - => techniques d'ordonnancement de tâches et de communications.

Caractéristiques d'un STR
1.1. Définitions...

- Quelques exemples :
  - Commande et contrôle de chaînes de production
  - guidage de systèmes mobiles (robotique…)
  - systèmes embarqués (avion, train, automobile…)
  - surveillance de réactions ou phénomènes physiques (nucléaire, chimie,…)
  - contrôle de malades et assistance d'opérations médicales
  - systèmes de communication et multimédia…
  - systèmes dédiés (conduite d'expérience scientifique, traitement du signal…)

Caractéristiques d'un STR
1.1. Définitions...

Quelques exemples

◆**Robot de production** : un robot, réalisant une activité spécifique (peinture, assemblage, tri) sur une chaîne de production, doit effectuer son travail en des temps fixés par la cadence de fabrication. S'il agît trop tôt ou trop tard, l'objet manufacturier traité sera détruit ou endommagé conduisant des **conséquences financières ou humaines graves** (oubli d'un ou plusieurs rivets sur un avion).

◆**Robot d'exploration** : ce robot doit se déplacer dans un environnement en principe non connu (zone radioactive après un accident, planète, épave sous la mer, …). Il est important qu'il puisse réagir aux obstacles fixes ou mobiles afin de ne pas **conduire à sa perte**.

◆**Téléphone mobile** : le système de contrôle-commande doit remplir plusieurs fonctions dont certaines ont des contraintes temporelles fortes pour avoir une bonne **qualité de service** (*quality of service*). Ainsi, la première fonction est de transmettre et de recevoir les signaux de la parole (577 µs de parole émises toutes les 4,6 ms et 577 µs de parole reçues toutes les 4,6 ms à des instants différents). En parallèle, il est nécessaire de localiser en permanence le relais le plus proche et donc de synchroniser les envois par rapport à cette distance (plus tôt si la distance augmente et plus tard si la distance diminue). Des messages de comptes rendus de la communication sont aussi émis avec une périodicité de plusieurs secondes. Les contraintes temporelles imposées au système doivent être imperceptibles à l'utilisateur.

Caractéristiques d'un STR
1.1. Définitions...

Quelques exemples

◆**Système de vidéo-conférence** : ce système doit permettre l'émission et la réception d'images numérisées à une cadence de 30 images par seconde pour avoir une bonne **qualité de service** (*quality of service*). Afin de minimiser le débit du réseau, une compression des images est effectuée. D'autre part la parole doit être aussi transmise; bien que correspondant à un débit d'information moindre, la régularité de la transmission (**gigue temporelle**) est nécessaire pour une reproduction correcte. De plus ce signal doit être synchronisé avec le flux d'images. L'ensemble de ces traitements (numérisations images et parole, transmission, réception, synchronisation, …) sont réalisés en cascade, mais avec une cohérence précise.

◆**Pilotage d'un procédé de fabrication** (fonderie, laminoir, four verrier, …) : la fabrication d'une bobine d'aluminium (laminage à froid) exige un contrôle en temps réel de la qualité (épaisseur et planéité). Cette vérification en production de la planéité nécessite une analyse fréquentielle (FFT) qui induit un coût important de traitement. Le système doit donc réaliser l'acquisition d'un grand nombre de mesures (246 Ko/s) et traiter ces données (moyenne, FFT, …) à la cadence de 4 ms . Ensuite, il affiche un compte-rendu sur l'écran de l'opérateur toutes les 200 ms et enfin imprime ces résultats détaillés toutes les 2 s. Un fonctionnement non correct de ce système de contrôle de la qualité peut avoir des **conséquences financières** importantes : production non conforme à la spécification demandée.

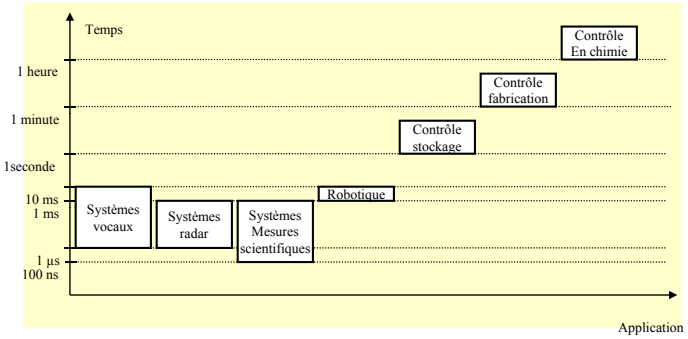
Caractéristiques d'un STR
1.1. Définitions...

Aspect temporel de la dynamique du procédé :

➡ dynamique du procédé très différente suivant l'application :

- milliseconde : systèmes radar, …
- seconde : systèmes de visualisation, …
- minutes : chaîne de fabrication
- heures : contrôle de réactions chimiques, …

Ordre de grandeur de la contrainte temporelle



Caractéristiques d'un STR
1.1. Définitions...

➢ Conséquences de la définition :

↳ Temps Réel ≠ aller vite

- temps de réaction court (1 ms) pour le contrôle d'un avion de combat
- temps de réaction moins court (10 ms) pour le contrôle d'un avion de transport civil
- temps de réaction moins court (1s) pour une IHM
- temps de réaction moins court (1mn) pour le contrôle d'un chaîne de production (lente)
- temps de réaction moins court (1h) pour le contrôle d'une réaction chimique (lente)
- temps de réaction de quelques heures pour l'établissement d'une prévision météorologique
- temps de réaction de quelques jours pour le calcul de la paie…

↳ => **On parlera de temps réel à chaque fois qu'il sera question de contraintes de temps et que ces dernières seront respectées :**

=> un résultat juste mais hors délai est un résultat faux

↳ => et parfois réagir trop tôt est aussi une faute (cf. contrôle d'un avion souple)

↳ **Temps Réel = Ponctualité**

Caractéristiques d'un STR
1.1. Définitions...

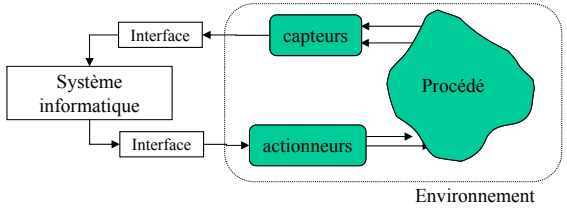
- Deux notions de criticité face au manquement d'une échéance
  - Contraintes temps réel **strictes** : le dépassement d'une échéance est catastrophique
    - **Contraintes de temps Fortes**
  - Contrainte temps réel **relatives** : le dépassement d'une échéance peut être toléré (dans une certaine mesure)
    - **Contraintes de temps Faible**
  - **=> Temps réel Dur / Temps réel Lâche !**
  - => dans le cas des systèmes 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...

Caractéristiques d'un STR
1.2. Un bref historique du temps réel...

- Fin des années 50…
  - première apparition de machines numériques dans des applications temps réel (système de commandes d'une unité de polymérisation d'une raffinerie Texaco)
- Années 60…
  - les systèmes temps réel s'étendent à des secteurs critiques tels que l'aérospatiale (système de contrôle de vol des capsules Apollo (NASA), système de commande de vol du Concorde (Sud Aviation)…)
- Années 70…
  - l'arrivée des mini-calculateurs accélère l'insertion d'outils informatisés dans les environnements temps réel (chaîne de production…)
- Depuis les années 80…
  - invasion de tous les secteurs industriels par l'informatique temps réel grâce à la micro-informatique, et au développement des techniques de communication
  - => systèmes temps réel distribués
  - => explosion de l'informatique embarquée (aérospatial, nucléaire, télécom, marine, automobile…)

Caractéristiques d'un STR
1.3. Archi. générale et modes de fonctionnement

- Principe :
  - Quelle que soit la nature et la complexité du système, on décompose un système temps réel en :
    - le système contrôlé
    - le système de contrôle



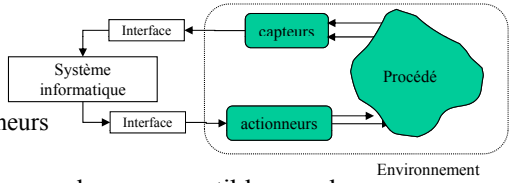
  - **<u>Le système contrôlé</u>** = environnement (procédé) équipé d'une instrumentation qui réalise l'interface avec le système de contrôle
  - **<u>Le système de contrôle</u>** = éléments matériels (microprocesseurs…) et logiciels dont la mission est d'agir sur le procédé via les actionneurs en fonction de l'état de ce procédé indiqué par les capteurs de manière maintenir ou conduire le procédé dans un état donné

Caractéristiques d'un STR
1.3. Archi. générale et modes de fonctionnement

- Exemple de systèmes contrôlés :
  - équipements isolés :
    - robot, machine à laver, caméra, radar, gouvernes d'un avion, moteur…
  - systèmes complexes :
    - chaîne de production, avion global, centrale nucléaire, contrôle aérien, surveillance médicale...

Caractéristiques d'un STR
1.3. Archi. générale et modes de fonctionnement

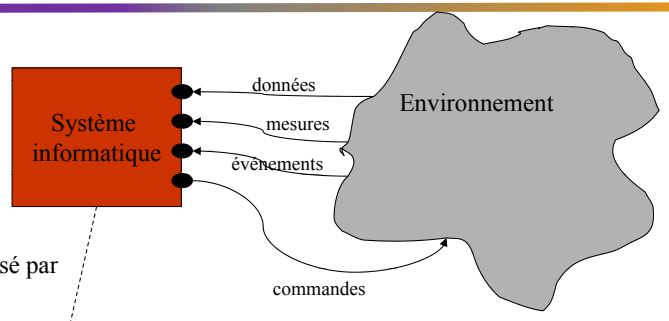
- Principe de fonctionnement :
  - Interaction :
    - via les mesures issues des capteurs,
    - et les commandes envoyées aux actionneurs



  - Acquisition des mesures périodiques à une cadence compatible avec les dynamiques du procédé
  - élaboration et envoi des commandes dans un laps de temps compatible avec les dynamiques du procédé

- Le double rôle du système de contrôle
  - observateur : acquisition des mesures
  - contrôleur : mise en œuvre des lois de commandes et émission des commandes vers des actionneurs spécifiques

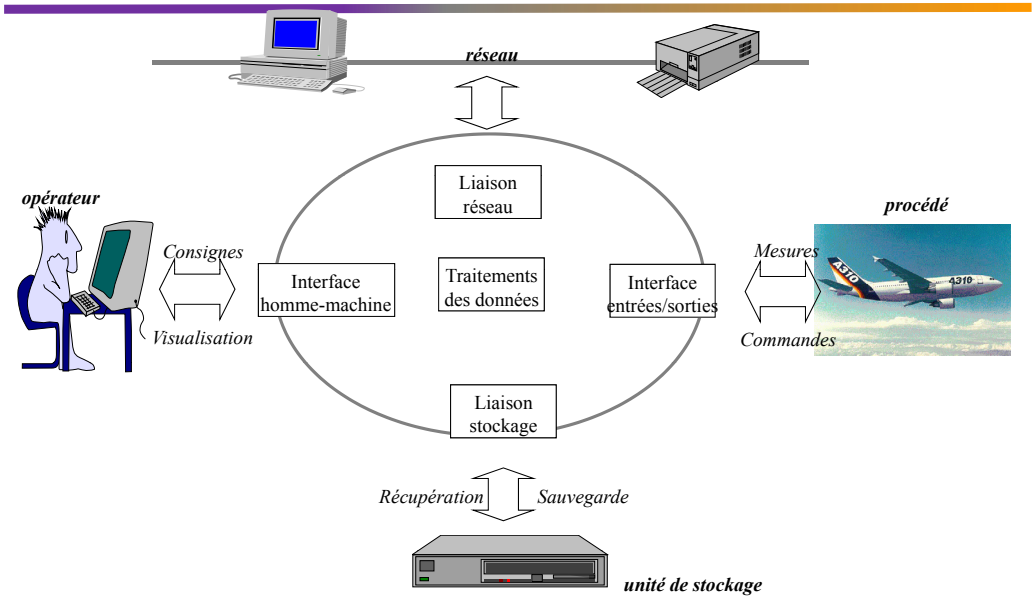
Caractéristiques d'un STR
1.3. Archi. générale et modes de fonctionnement



Peut être réalisé par

- un automate programmable
- un circuit spécifique (ASIC)
- un calculateur (monoprocesseur, microcontroleur)
- un système multiprocesseur à mémoire commune
- des systèmes répartis
- …

Caractéristiques d'un STR
1.3. Archi. générale et modes de fonctionnement



Caractéristiques d'un STR
1.3. Archi. générale et modes de fonctionnement

- Fonctionnement général : boucle infinie

```
Tant que TOUJOURS faire
  • Acquisition des entrées (données capteurs, mesures…)
  • Calcul des ordres à envoyer au procédé
  • Émission des ordres
Fin tant que
```

- Mais, deux modes de fonctionnement :
  - fonctionnement **cyclique** (time driven ou système "synchrone" (mais pas le même "synchrone" que les langages "synchrones"))
  - fonctionnement **événementiel** (event driven)
  - => fonctionnement mixte : à base de traitements périodiques et apériodiques (déclenchés sur événements)

Caractéristiques d'un STR
1.3. Archi. générale et modes de fonctionnement

- Fonctionnement Cyclique :
  - scrutation d'une mémoire d'entrée périodiquement (polling)
  - => échantillonnage des entrées sur l'horloge du système
  - => activation du système à chaque top d'horloge
    - A chaque top d'horloge faire
      - Lecture de la mémoire des entrées
      - Calcul des ordres à envoyer au procédé
      - Émission des ordres
    - Fin

  - **Mais :**
  - système peu "réactif" si l'environnement produit des informations à des fréquences différentes
  - => oblige à prévoir toutes les réactions du système dans la même boucle
    - => problème de performance
  - => ou oblige à imbriquer des boucles de fréquences multiples
    - => difficultés de réalisation, de lisibilité du code, d'évolution

Caractéristiques d'un STR
1.3. Archi. générale et modes de fonctionnement

- Fonctionnement Evénementiel :
  - activation du système à chaque événement (=> notion d'interruption)
    - A chaque interruption faire
      - Lecture de l'information arrivée
      - activation du traitement correspondant
      - Émission des ordres issus de ce traitement
    - Fin

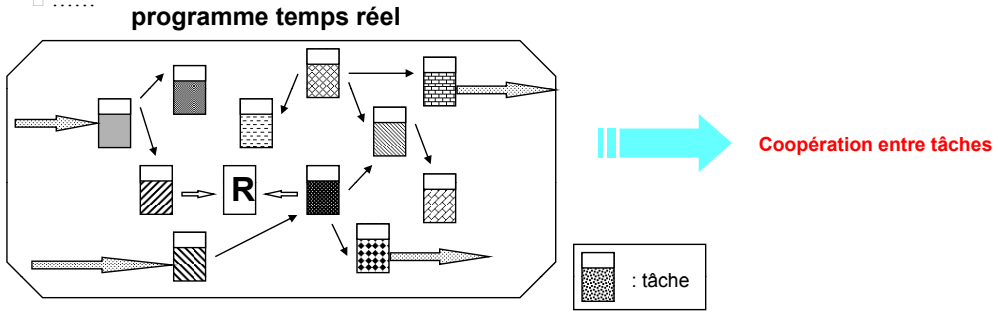
  - **Mais :** que faire si une interruption survient alors que le système est en train de traiter une interruption précédente
  - => notion de priorité des interruptions
  - => notion de "tâche" associée à une ou plusieurs interruptions
  - => mécanisme de préemption et de reprise de tâche
  - => gestion de l'exécution concurrente des tâches (ordonnancement)

- => Un système temps réel est souvent un système multitâche incluant un gestionnaire de tâches (Ordonnanceur)

Caractéristiques d'un STR
1.3. Archi. générale et modes de fonctionnement

=> Pour des raisons de facilité de conception, mise en œuvre et d'évolutivité, une application temps réel est un système multitâche :

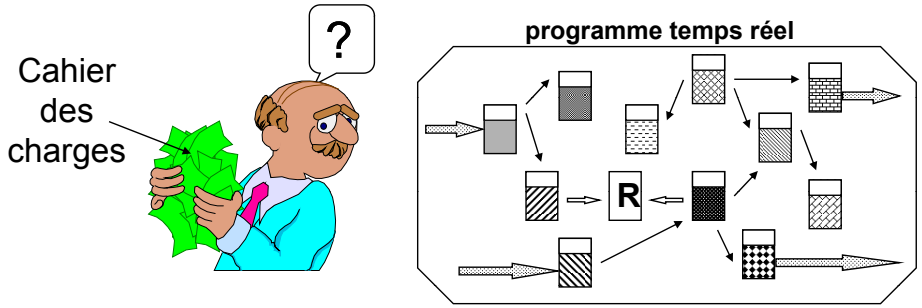
- tâche associée à un ou des événements
- tâche associée à une ou des réactions
- tâche associée à une entité externe à contrôler
- tâche associée à un traitement particulier
- ……



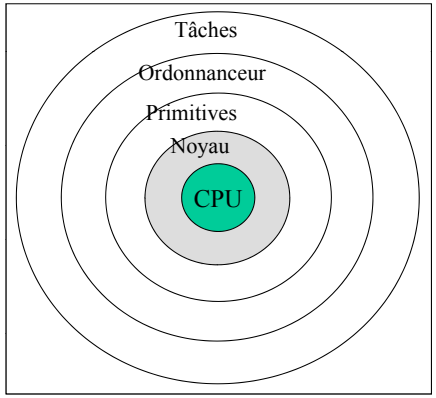
Caractéristiques d'un STR
1.3. Archi. générale et modes de fonctionnement

=> Un ordonnancement :

La partie logicielle d'une application temps réel peut être vue comme un ensemble de tâches synchronisées, communicantes et partageant des ressources critiques. Le rôle essentiel du système informatique temps réel est donc de gérer l'enchaînement et la concurrence des tâches en optimisant l'occupation de l'unité centrale : fonction d'**ordonnancement**.



- Organisation générale d'un OS temps réel



↳ Noyau + primitives (de services) + ordonnanceur = système d'exploitation

- Notion d'OS temps réel:
  - ↳ **Système conduit par les données (Data Driven)**
    - **gérer un grand nombre d'E/S en parallèle,**
    - **gérer et ordonnancer les tâches,**
    - **communiquer, synchroniser,**
    - **interruption et priorité sur les interruptions**
    - **gestion du temps, datation des informations,**
    - **minimiser l'effet de défaillance,**
    - **sécurité.**

- Les différents aspects d'un OS temps réel:
  - **Noyau ou moniteur temps réel (Kernel)**
    - **RTC - 3 à 4 Ko, 27 fonctions (Norme SCEPTRE)**
  - **Exécutif temps réel**
    - **VRTX (SE élémentaire)**
  - **Système d'exploitation temps réel**
    - **OS9 (SETR) (Norme POSIX 1003.1b)**

- NB : temps physique continu ≠ temps informatique discrétisé
- ⇒ plusieurs représentations du temps non exclusives :
  - ↳ référence ponctuelle (date = point sur l'échelle des temps)
    - ⇒ décrire les dates, événements qui déterminent le déroulement de certaines activités (approche *synchrone*)
  - ↳ notion de durée ou d'intervalles de temps
    - ⇒ décrire, modéliser les recouvrements et le parallélisme de certaines activités (approche *asynchrone*)
- Remarques :
  - ↳ intervalle de durée nulle = point
  - ↳ horloge = référence temporelle absolue
  - ↳ ☹ pb de la synchronisation des horloges

Caractéristiques d'un STR
1.4. Contraintes d 'exécution des tâches

- Contraintes de niveau système :
- *des contraintes d'environnement, masse, volume, consommation, performances, ...*
- *des contraintes de bon fonctionnement :* Fiabilité
  - Objectif → raisons économiques (maîtrise coûts d'exploitation par la fiabilité, maintenabilité et testabilité) et de sécurité (certificabilité)
  - Prise en compte de contraintes de SdF ⇒ fiabilité, qualité, conception de systèmes tolérants les fautes (ressources redondantes, reconfigurations, reprises temporelles des traitements, …)
- *Des contraintes de réalisation*
  - Complexité, contraintes de SdF ⇒ approche qualité par l'utilisation systématique de méthodologies rigoureuses pour la conception, réalisation, validation, évolution et maintenance des systèmes (logiciels) : AGL

Caractéristiques d'un STR
1.4. Contraintes d 'exécution des tâches

- *Des contraintes de prédictibilité:* Prévisibilité
  - Prédictibilité : qualité d'un système qui, à partir de la connaissance de son état présent (fonctionnalités, performance, …) permet d'assurer la connaissance de son comportement dans le futur et garantit la maîtrise de son fonctionnement en cas de défaillance prévue ou non prévue
  - Possibilité de décrire le comportement dans le temps par un modèle statique fonctionnalités-performances-contraintes ⇒ rend prédictible le comportement futur

note: sous contraintes d'hypothèses simplificatrices qui restreignent la garantie de bon fonctionnement à ce domaine de validité…

Caractéristiques d'un STR
1.4. Contraintes d 'exécution des tâches

- Contraintes de niveau actions :
- *Des contraintes de communication*
  - Les tâches concourent à contrôler/commander un procédé ⇒ elles interagissent (émission/réception de signaux de synchronisation + informations) :
    - boite aux lettres en mémoire commune (système centralisé)
    - transfert par messages (système distribué)
      ⇒ délais de transmission non négligeables (support, absence de cohérence d'horloges locales, difficulté de construire un schéma de contrôle global)

Caractéristiques d'un STR
1.4. Contraintes d 'exécution des tâches

- *Des contraintes de criticité* : Urgence
  - Sensibilité de l'action (mesurée par les conséquences de l'occurrence d'une faute) sur le bon fonctionnement du système
  - Exemple : non-respect des dates de réveil et échéances ⇒ conséquences dont la sévérité dépend de la tâche fautive
    (perte acceptable de performance → catastrophique)
  - A chaque tâche est affecté un niveau de criticité qui conduira à privilégier l'exécution des tâches de grande importance

Caractéristiques d'un STR
1.4. Contraintes d 'exécution des tâches

- *Des contraintes de préséance*
  - hiérarchie entre activités selon leurs importances relatives (ex. détection/traitement des fautes, activités brèves, urgentes, liées à certains événements, …)
    - **⇒ concept de priorité**

- *Des contraintes de préemptiblité*
  - une activité peut être interrompue ou pas par une activité de plus forte préséance
    - **⇒ retrait des ressources (processeur) pour les affecter à la nouvelle activité (reprise ultérieure quand les circonstances le permettront)**

Caractéristiques d'un STR
1.4. Contraintes d 'exécution des tâches

- *Des contraintes d'activation*
  - L'exécution d'une activité peut commencer au plus tôt après la réalisation de la *condition d'activation* (date, occurrence d'un événement externe, interne (transition d'état), temporel (fin de délai, début de période si activité périodique) ou d'autre nature (disponibilité de ressources, fin d'une autre activité, condition de synchronisation)
    - Possibilité de contraindre la condition dans le temps (prise en compte entre 2 dates ou pour une durée limitée (time-out)
    - 1 ou plusieurs conditions élémentaires en OU ou ET

Caractéristiques d'un STR
1.4. Contraintes d 'exécution des tâches

- *Des contraintes d'implantation (placement: cas des systèmes distribués)*
  - Portent sur l'identité du système distribué sur lequel une tâche est autorisée à s'exécuter (souvent calqué sur la répartition géographique) : prendre en compte pour évaluer la charge de traitement de chaque nœud, …
    - **⇒ activités *voisines* ou *distantes***
    - **Problème crucial en cas de redondance matérielle/logicielle pour la conception de systèmes sûrs de fonctionnement (tolérer la panne d'un/plusieurs nœuds et le dépassement d'échéance par une/plusieurs tâches)**

Caractéristiques d'un STR
1.4. Contraintes d 'exécution des tâches

- *Différents types de contraintes temporelles selon que l'activité est :*
  - périodique (cyclique) : activité soit réalisée une fois par période T, soit 2 exécutions consécutives séparées par une durée T
    - ex: acquisition d'informations
  - sporadique (asynchrone) : réalisée à la demande ou sous des conditions d'activation non périodiques
    - ex: événements processus, détection faute, intervention humaine

Caractéristiques d'un STR
1.4. Contraintes d 'exécution des tâches

- *Contraintes temporelles :*
  - date de début au plus tôt, date de début au plus tard

    La date de début doit être :
    - postérieure à la date de réalisation de la condition d'activation et à la date de début au plus tôt
    - antérieure à la date de début au plus tard
  - date de fin au plus tôt, date de fin au plus tard

    La date de fin d'activité doit être :
    - postérieure à la date de fin au plus tôt
    - antérieure à la date de fin au plus tard

Caractéristiques d'un STR
1.4. Contraintes d 'exécution des tâches

- date de fin au plus tard = date de forclusion ou deadline
- délai de forclusion (délai de cohérence) :
  - durée qui s'écoule depuis la date de réalisation de la condition d'activation jusqu'à la date de forclusion
  - L'activité doit impérativement être terminée avant la fin du délai de forclusion.
  - A chaque activité est associée une durée d'exécution.
  - L'exécution d'une activité peut être :
    - continue (non interrompue par préemption) : *connexe*
    - non-connexe : les mêmes contraintes de début/fin d'activité sont applicables

Caractéristiques d'un STR
1.4. Contraintes d 'exécution des tâches

- Expression des contraintes temporelles :
  - exigence de commencer l'exécution d'une tâche après avoir satisfait des conditions nécessaires de départ
  - exigence de terminer son exécution avant une échéance
- La date de début d'exécution au plus tôt (date de réveil) découle directement ou indirectement de l'occurrence de signaux émis ou prélevés sur l'environnement
- La date de fin d'exécution au plus tard (échéance) est directement ou indirectement liée aux dynamiques de l'environnement
- Toute tâche caractérisée par une date de réveil et (ou) une échéance est en général qualifiée de tâche à date critique

Caractéristiques d'un STR
1.4. Contraintes d 'exécution des tâches

- Déterminisme :
  - **Connaître exactement l'instant de début et de fin d'un traitement.**
    - ***Niveau Processeur***
    - ***Niveau Executif***
    - ***Niveau Utilisateur***
- Le problème du non-déterminisme :
  - Non déterminisme = caractéristique importante car pose de sérieux problèmes pour s 'assurer de la maîtrise du comportement du système
    - **Causes de non-déterminisme :**
      - » ***Charge du système***
      - » ***E/S***
      - » ***Interruption Externe / Interne (exception)***
  - Séquences d 'actions interrompues par des événements externes asynchrones à des instants et dans un ordre imprévisibles ⇒ ordre de traitement des séquences imprévisible !

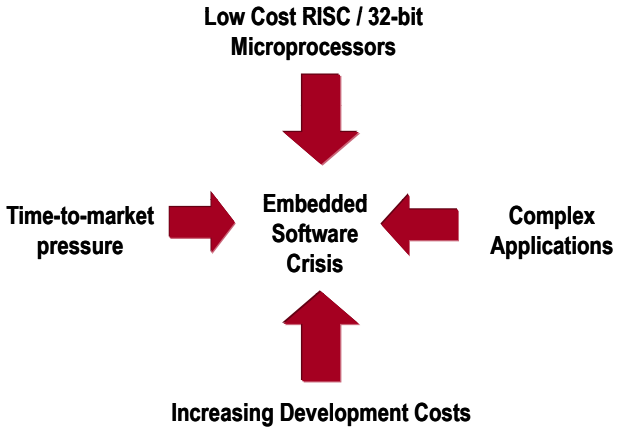
Caractéristiques d'un STR
1.5. Limites des systèmes classiques

- Limite des systèmes classiques pour le T.R.
  - Durée de traitement de l 'information non nulle,
  - Favorisent l'interface Homme - Machine plutôt que l 'interface Machine - Capteur,
  - Mécanismes concernés :
    - **Politique d 'ordonnancement,**
    - **Accès aux ressources partagées, Synchronisation, Communication,**
    - **Gestion des interruptions**
    - **Gestion de la mémoire**
    - **Gestion du temps.**

Caractéristiques d'un STR
1.6. Généralité sur les OS TR

◆ **Objectifs des RTOS :**

- fournir du logiciel de qualité et des services pour que l'utilisateur arrive à créer des applications embarquées rapidement, à faible côut et avec un risque minimum.



Ajouté à cela :

- de nouvelles technologies
- de nouvelles applications
- l'explosion des semiconducteurs
- l'Internet

Caractéristiques d'un STR
1.6. Généralité sur les OS TR

Les principales caractéristiques des noyaux temps réel

- **Conformité à une norme** ou pseudo-norme (POSIX, projet Sceptre)
- **Compacité** (pour les applications embarquées)
- **Environnement cibles** (microprocesseurs, architecture, …)
- **Environnement hôtes** (type OS)
- **Outils d'aide au développement** (debug, analyse en ligne, …)
- **Primitives temps réel** (liste de tous les services fournis)
- **Caractéristiques de l'ordonnanceur** (politiques d'ordonnancement)
- **Caractéristiques temporelles**
  - ***temps de masquage des interruptions*** (*interrupt latency*) : temps pendant lequel les interruptions sont masquées et ne peuvent donc pas être prises en compte (exécution de primitives atomiques, manipulation des structures critiques, …)
  - ***temps de réponse*** (*task response time*) : temps entre l'occurrence d'une interruption et l'exécution de la tâche réveillée
  - ***temps de retard de l'ordonnanceur*** (*preemptive latency*) : temps maximal pendant lequel le noyau peut retarder l'ordonnanceur.

Caractéristiques d'un STR
1.6. Généralité sur les OS TR

- Deux grandes catégories d'O.S. temps réel :
  - => les OS Temps Réel d'origines (RTC, WxWorks, Tornado, QNX, …)
    - offrent une gestion fine des priorités
    - offrent des primitives système rapides, en temps borné (gestion des interruptions, des sémaphores…)
    - pas de mémoire virtuelle, mais verrouillage de pages en mémoire centrale
    - minimisation de l' "overhead" (le temps pris par le système pour s'exécuter et se gérer lui-même)
    - => la solution au Temps Réel Dur

Caractéristiques d'un STR
1.6. Généralité sur les OS TR

Comparaison de noyaux temps réel du marché

Nom	Société	Env. cible	Env. hôte	Compacité	Tps Rép Int	Tps Com. Context	Norme	Ordo.	Application
RTC Real Time Craft	GSI-Tecsi (France)	- 680x0 - Intel x86, Pentium - NS - multiprocesseur	- Windows - Unix	4 Ko.	15 µs	8 µs	Sceptre	- 65332 niv. prio. - tourniquet	- automobile, - etc.
pSOS+	Integrated Systems Inc. (USA)	- 680x0 - PowerPC - Intel x86, Pentium - i960 - multiprocesseur	- Windows - Unix	10 Ko. (PSOSelect : 1,8 Ko)	7 µs	19 µs	≈ Posix	- 255 niv. prio. - tourniquet	- console de jeu (SEGA), - etc.
VRTXsa Versatile Real Time eXecutive	Integrated Systems Inc. (USA)	- Motorola - Intel	- Unix	13 Ko. (VRTXmc : 4 Ko)	10 µs	15 µs	≈ Posix	- 255 niv. prio. - tourniquet - héritage de priorité	- téléphonie - etc.
VxWORKS	Wind River Systems (USA)	- Motorola - Intel - RSx000	- Windows - Unix	12 Ko.	8 µs	16 µs	Posix	- 255 niv. prio. - tourniquet - héritage de priorité	- automobile - espace (NASA) - etc.
LynxOS	Lynx Real-Time System (USA)	- Motorola - Intel - RSx000	- Windows - Unix	-			Posix	- 255 niv. prio. - tourniquet - héritage de priorité avec priorité plafond	- imprimante (Xerox) - espace (NASA) - etc.
QNX	Inforange ou QNX Systems (Canada)	- Intel	- Windows	-			Posix	- 16 niv. prio. - tourniquet	- tunnel sous la Manche - etc.
CHORUS	Chorus (France)	- Motorola - Intel	- Unix	-			Posix	- 256 niv. prio. - tourniquet	
OS9	Microware Systems (USA)	- Motorola - Intel	- Windows - Unix	- 16 Ko	11 µs	140 µs		- 65535 niv. prio. - tourniquet - vieillissement	- radio-téléphone - CD interactif - etc.
Windows CE	Microsoft (USA)	- Motorola - Intel	- Windows	200 Ko à 32 Mo	10 µs	18 µs	Posix	- 32 niv. prio. - tourniquet	Industrie

Caractéristiques d'un STR
1.6. Généralité sur les OS TR

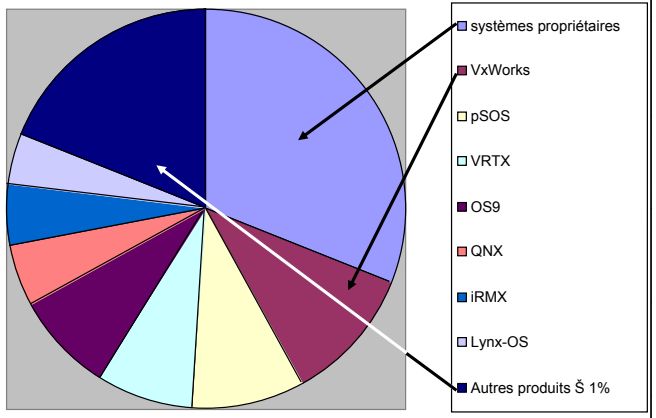
- Deux grandes catégories d'O.S. temps réel :
  - => les O.S. classiques étendus pour le temps réel (Windows CE, Unix…)
    - permettent le développement concurrent d'applications temps réel et non temps réel dans un environnement standard et confortable
    - Mais il a fallu
      - Revoir les politiques d'ordonnancement
      - Renforcer la notion de préemption
      - Définir des primitives systèmes réentrantes
      - Définir la notion de processus léger (pour faciliter la préemption avec sauvegarde de contexte puis la reprise avec restitution de contexte)
    - => modifications importantes et complexes
    - => pour le temps réel lâche

Caractéristiques d'un STR
1.6. Généralité sur les OS TR

Situation de l'offre industrielle des noyaux temps réel

systèmes propriétaires	31
VxWorks	11
pSOS	9
VRTX	8
OS9	8
QNX	5
iRMX	5
Lynx-OS	4
Autres produits <1%	19

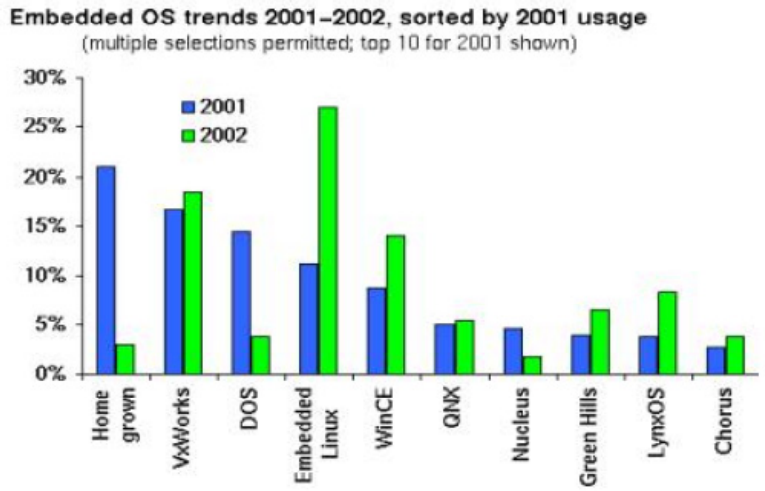
Étude de Embedded Systems Research (marché USA - 1995)



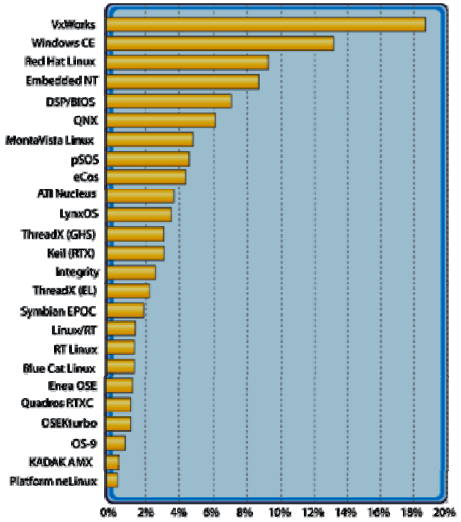
- Pas de prédominance d'un système
- système propriétaires :
  - coût (Licence)
  - techniques (adaptation aux besoins)
  - stratégie (maîtrise)

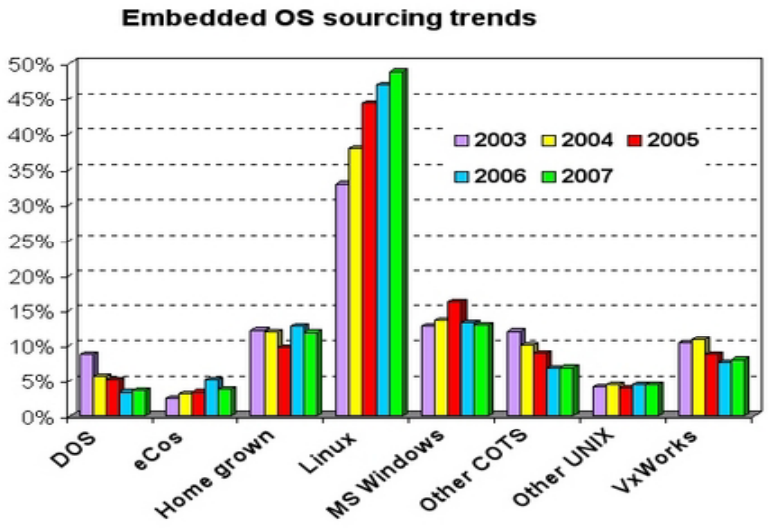
Caractéristiques d'un STR
1.6. Généralité sur les OS TR

Situation de l'utilisation des noyaux temps réel



Répartition des OS en 2005





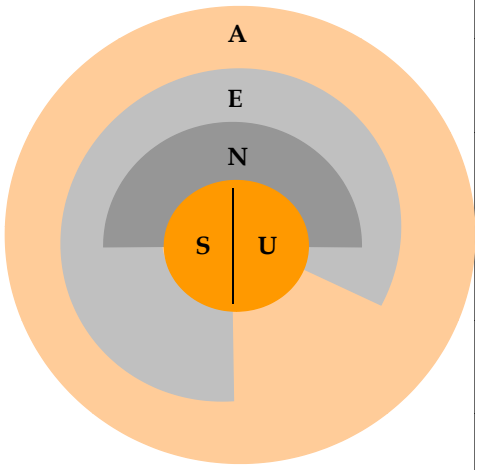
Snapshot of the embedded Linux market -- April, 2007

- Objectifs :
  - Problème de coopération entre activités parallèles.
  - Doit permettre de construire de façon procédurale la plupart des mécanismes de coopération à partir d'un nombre d'objets restreints et d'opérations élémentaires.
  - Le noyau SCEPTRE n'est pas portable.
  - Les applications utilisant SCEPTRE doivent assurer une portabilité maximale.
  - Le noyau distingue trois classes de fonctions :
    - » Exclusion Mutuelle, Signalisation, Communication
  - 7 points de passages obligés : Adéquation aux besoins, Sécurité, Diminution des coûts et délais de réalisation et maintenance, **Performance**, **Portabilité**, Adaptabilité, Modularité.

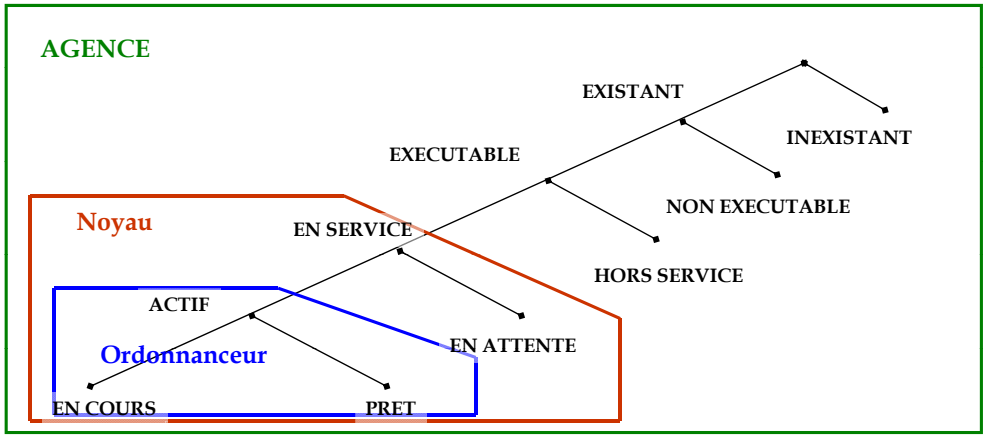
- Historique :
  - Standardisation du Cœur des Exécutifs des Produits Temps Réels Européens. (fin des années 70)
    - Fonctions de bas niveau
    - Définition des termes du « temps réel »
- Approche Temps Réel de SCEPTRE:
  - Le noyau se limite aux éléments suivants :
    - Gestion des tâches
    - Signalisation
    - Exclusion Mutuelle
    - Communication entre tâches
    - Communication avec l'extérieur
    - Traitement des situations d'exceptions

- Ensemble des services d'un exécutif TR. (POSIX)
  - Communication
  - Synchronisation
  - Gestion et ordonnancement des tâches
  - Gestion de la mémoire
  - Gestion des interruptions et entrées/sorties physiques
  - Entrées/sorties logiques et gestion des périphériques
  - Gestion des fichiers (logique et physique)
  - Gestion des programmes
  - Gestion des travaux et des transactions
  - Traitement des erreurs et des exceptions
  - Gestion du temps

- Règles de visibilité de SCEPTRE :
  - Niveau Application
    - Mode Utilisateur + Exécutif
  - Niveau Exécutif
    - Mode Utilisateur + Superviseur + Noyau
  - Niveau Noyau



- Les groupes d'états ou états d'une tâche SCEPTRE:



- Les groupes d'états ou états d'une tâche SCEPTRE:
  - **INEXISTANT** : Il n'y a pas de descripteur associé à la tâche.
  - **EXISTANT** : La tâche possède un descripteur défini.
  - **NON-EXECUTABLE :** La tâche possède un descripteur défini, mais elle ne peut ni démarrer son exécution, ni la continuer.
  - **EXECUTABLE :** La tâche possède un descripteur défini et elle peut s'exécuter.
  - **HORS-SERVICE** : La tâche est EXECUTABLE, mais on n'a pas encore demandé son démarrage ou bien son exécution est terminée.
  - **EN-SERVICE** : La tâche est EXECUTABLE, elle a commencé son exécution et ne l'a pas terminée.
  - **EN-ATTENTE** : La tâche est EN-SERVICE et attend qu'une condition explicite soit satisfaite pour continuer son exécution.
  - **ACTIF** : La tâche est EN-SERVICE et n'attend aucune condition explicite, sauf peut-être qu'un processeur soit libre pour pouvoir s'exécuter.
  - **EN-COURS** : La tâche est ACTIVE et elle dispose d'un processeur pour s'exécuter.

Le Projet SCEPTRE
2.1. Présentation

TYPES	SIGNIFICATION	MANIPULE PAR
TACHE	***Un objet de type TACHE désigne une tâche gérée par le noyau***	Opérations élémentaires de gestion des tâches
EVENEMENT	***Un objet de type EVENEMENT mémorise le signal marquant qu'une condition est devenue vraie***	Opérations élémentaires de signalisation
REGION	***Un objet de type REGION contient une marque mémorisant qu'un objet partagé est (ou non) possédé par une tâche gérée par le noyau***	Opérations élémentaires d'exclusion mutuelle
FILE	***Un objet de type FILE range des informations transitoires et les restitue suivant l'ordre de leur arrivée dans la file (premier entré, premier sorti)***	Opérations élémentaires de communication

Le Projet SCEPTRE
2.2. Les Opérateurs T. R. de SCEPTRE

- Les primitives de SCEPTRE:
  - 16 Primitives, 3 prédicats,
  - Propriétés d'exclusivité pour les opérations du noyau,
  - Propriété d'atomicité (ou indivisibilité),
  - Propriété de sécurité.
- La gestion des Tâches :
  - 4 Opérations élémentaires dans le noyau.
  - Opérations *ARRETER - CONTINUER* pour Exceptions.
  - Pas de destruction ni de création dans le noyau.
    - » Coûteux en Temps.

Le Projet SCEPTRE
2.2. Les Opérateurs T. R. de SCEPTRE

- La signalisation :
  - Les événements
    - Tout événement est associé à une tâche.
    - Il peut prendre 2 états : ***Arrivé*** ou ***Non - Arrivé.***
- La communication :
  - Les files
    - Files de tâches ou files de Messages.
    - FIFO.
    - Bornées ou non.

Le Projet SCEPTRE
2.2. Les Opérateurs T. R. de SCEPTRE

- Les primitives définies par SCEPTRE

CATEGORIE	OPERATION ELEMENTAIRE	PARAMETRES	FONCTION
GESTION DES TACHES	DEMARRER	TACHE	*Lancer l'exécution de la tâche*
	ARRETER	TACHE	*Arrêter la tâche*
	CONTINUER	TACHE	*Continuer l'exécution de la tâche*
	SE TERMINER		*Terminer l'exécution de la tâche qui l'exécute*
	CHANGER-PRIORITE	TACHE, nouvelle priorité	*Donner à la tâche la nouvelle priorité*
	ETAT	TACHE	*Fournir l'état de la tâche*
	PRIORITE	TACHE	*Fournir la priorité de la tâche*
	TACHE COURANTE		*Fournir l'identité de la tâche qui l'exécute*
	STATUT		*Fournir le statut de l'opération élémentaire qui vient d'être exécutée par la tâche*
SIGNALISATION	SIGNALER	EVENEMENT, TACHE	*Signaler à la tâche que l'événement est arrivé*
	ATTENDRE	Liste d'EVENEMENTS	*Attendre que l'un (au moins) des événements de la liste soit arrivé*
	ARRIVE	Liste d'EVENEMENTS	*Prédicat qui est vrai si tous les événements de la liste sont arrivés*
	EFFACER	Liste d'EVENEMENTS	*Remettre les événements de la liste dans l'état non arrivé*

Les primitives définies par SCEPTRE

CATEGORIE	OPERATION ELEMENTAIRE	PARAMETRES	FONCTION
COMMUNICATION ENTRE TACHES	ENVOYER	ELEMENT, FILE	*Envoyer l'élément à la queue dans la file*
	RETIRER	ELEMENT, FILE	*Retirer le premier élément de la file*
	VIDE	FILE	*Prédicat qui est vrai si la file est vide*
	PLEINE	FILE	*Prédicat qui est vrai si la file est pleine*
EXCLUSION MUTUELLE	ENTRER	REGION	*Demander la propriété exclusive de la région*
	SORTIR	REGION	*Relâcher la propriété exclusive de la région*

Sémantique des primitives de Gestion de tâches

DEMARRER (tâche)

Sémantique : La tâche passe de l'état HORS-SERVICE à l'état EN-SERVICE
Etat initial : HORS-SERVICE
Etat final : EN-SERVICE
Cas d'erreur : Si la tâche n'est pas dans l'état initial correct, le statut de la tâche appelante est positionné.

ARRETER (tâche)

Sémantique : La tâche est mise dans l'état HORS-SERVICE
Etat initial : EN-SERVICE
Etat final : HORS-SERVICE
Remarque : Pour toute réactivation ultérieure de la tâche, on doit utiliser l'opération CONTINUER.

CONTINUER (tâche)

Sémantique : Identique à celle de DEMARRER
Etat initial : HORS-SERVICE
Etat final : EN-SERVICE
Remarque : Les implémentations de DEMARRER et CONTINUER peuvent être différentes.

SE TERMINER (tâche)

Sémantique : La tâche courant est retirée à son processeur. Elle est mise dans l'état HORS-SERVICE et l'ordonnanceur est invoqué.
Etat initial : EN-COURS
Etat final : HORS-SERVICE
Remarque : Pour toute réactivation ultérieure de cette tâche, on doit utiliser l'opération DEMARRER.

Sémantique des primitives de Gestion de tâches

CHANGER-PRIORITE (tâche, nouvelle priorité)
Sémantique : L'opération CHANGER-PRIORITE donne à la priorité de tâche la valeur nouvelle priorité. Ceci permet de réaliser la modification de sa propre priorité que celle d'une autre tâche.

Etat initial :	EN-COURS	EN-ATTENTE	Prêt	HORS-SERVICE
Etat final :	PRET ou EN-COURS	EN-ATTENTE	PRET ou EN-COURS	HORS-SERVICE

Remarque : Cette opération peut induire une préemption du processeur de la tâche qui l'invoque et son passage à l'état prêt.

TACHE COURANTE (tâche)

Sémantique : Cette fonction permet à un programme de connaître l'identité de la tâche qui exécute ce programme. Elle fournit en résultat le nom de la tâche qui l'exécute.

PRIORITE (tâche)
Sémantique : Cette fonction fournit en résultat la priorité de la tâche.

ETAT (tâche)
Sémantique : Fournit l'état de la tâche.

STATUT (tâche)
Sémantique : Fournit le compte-rendu de la dernière opération élémentaire du noyau exécutée par la tâche.

Sémantique des primitives de Signalisation:

ARRIVE ([e1, … , en])
Le prédicat est vrai si tous les événements de la liste sont arrivés.

SIGNALER (e,t)
Sémantique : Quel que soit l'état initial de l'événement e, il est mis à l'état arrivé.

EFFACER ([e1, … , en])
Sémantique : Les événements de la liste sont mis à l'état non - arrivé.

ATTENDRE ([e1, … , en])
Sémantique : La tâche courante se mets en attente de l'arrivée d'au moins un événement de la liste.

➢Sémantique des primitives de Communication:

ENVOYER (élément, file)

Sémantique : Si File est non bornée ou bornée et non pleine, on range élément à la suite des éléments déjà rangés dans File.

Etat initial : File peut être vide ou pleine dans le cas des files bornées.

Cas d'erreur : Si File est bornée et pleine.

RETIRER (élément, file)

Sémantique : Lorsque la File n 'est pas vide, on retire le premier élément et on le retourne dans élément.

Etat initial : File peut être vide ou non vide.

Cas d'erreur : Si File est vide.

VIDE (file)

Ce prédicat est vrai lorsque la file est ***Vide.***

PLEINE (file)

Ce prédicat est vrai lorsque la file est ***Pleine*** (dans le cas d'une file bornée).

➢ L'exclusion mutuelle :

↳Les sections critiques

- Lorsque plusieurs tâches exécutent concurremment l'opération d'entrée en SC, seule l'une d'elles peut terminer cette opération.
  - Accès exclusif
- Les autres tâches doivent attendre que la tâche propriétaire ait exécuter une opération de sortie de SC.
- Pas de directive sur l'ordre d'attribution ni sur le type d'attente.

↳Le concept de Région

- Base à des services de plus haut niveau (sémaphores,…).
- Aucun passage à l'état *EN-ATTENTE* dans une Région.
- Pas de préemption de la tâche courante tant qu'elle exécute des instructions d'une SC protégée par une Région.
- L'objet Région peut prendre 2 états grâce à 2 opérations :
  - ***Libre*** ou ***Occupé*** / *ENTRER* - *SORTIR*
- Les opérations entraînant la préemption (*SIGNALER, DEMARRER, CHANGER-PRIORITE et ARRETER*) sont correctes dans une SC, mais ne peuvent invoquer l'algo de choix jusqu'à l'opération *SORTIR*.
- Les opérations *ATTENDRE* et *TERMINER* sont interdites.

➢Sémantique des primitives d'Exclusion Mutuelle:

ENTRER (Région)

Etat initial : La Région est LIBRE.

Etat final : Le mode d 'exécution de la tâche courante devient non préemptible pour l'ordonnanceur des tâches différées. La région est OCCUPEE.

Remarque : La région doit être libre lors du démarrage de l'exécution.

SORTIR (Région)

Sémantique : Lorsque le mode d 'exécution redevient préemptible, l 'ordonnanceur est invoqué.

Etat initial : La région est OCCUPEE. Le mode d 'exécution de la tâche courante devient non préemptible pour l'ordonnanceur des tâches différées.

Etat final : La Région est LIBRE.

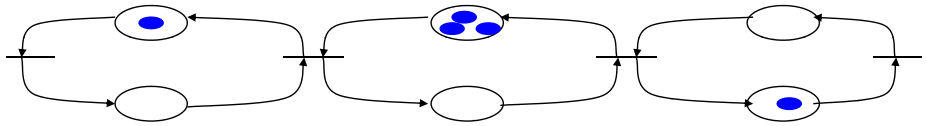
Remarque : Le mode d'exécution de la tâche courante après l'opération SORTIR est celui qu'elle avait avant la dernière exécution de l'opération ENTRER.

Le Projet SCEPTRE
2.2. Les Opérateurs T. R. de SCEPTRE

- Pseudo code des opérations sur région:
  - **Entrer (Région)**
    - Masquer les interruptions
    - ***Si*** REGION occupée ***alors***
      - Retourner un compte rendu d'erreur dans STATUT
    - ***Sinon***
      - Réserver REGION pour TACHE-COURANTE
    - ***Finsi***
    - Incrémenter la variable COMPTE-REGION de la tâche.
    - Indiquer que l'algorithme de choix ne doit pas s'exécuter.
    - Démasquer les interruptions.
  - **SORTIR (Région)**
    - Masquer les interruptions.
    - Libérer la REGION occupée par la TACHE-COURANTE
    - Décrémenter la variable COMPTE-REGION si elle est positive.
    - ***Si*** COMPTE -REGION non nul ***alors***
      - Démasquer les interruptions.
      - Retourner à la procédure appelante
    - ***Sinon***
      - Indiquer que l'algorithme de choix peut s'exécuter
      - Démasquer les interruptions
      - Invoquer l'algorithme de choix
    - ***Finsi***

Le Projet SCEPTRE
2.3. Les Opérateurs « Haut Niveau » de SCEPTRE

- Le modèle Producteur-Consommateur:
  - Met en jeu une File ***F*** et un événement ***F-NON-VIDE.***
  - Pb avec file de type FIFO.



Producteur *Tampon* *Consommateur*

- Les sémaphores
  - Fonctions Prendre et Vendre utilisant les primitives *ENVOYER, EFFACER, ATTENDRE, ENTRER et SORTIR*.
  - Booléens
    - Deux états : Libre ou occupé.
      - Exclusion mutuelle
  - A compte
    - Ressource réentrante pas forcément critique.

Le Projet SCEPTRE
2.3. Les Opérateurs « Haut Niveau » de SCEPTRE

- Les « monitors »
  - Assimilable à un « sémaphore booléens conditionnel »
  - Notion de propriété exclusive du moniteur
  - Primitives associées :
    - Enter, Exit, Wait, Signal, Signal-exit

Le Projet SCEPTRE
2.3. Les Opérateurs « Haut Niveau » de SCEPTRE

- Les « control-queues »
  - Objet de synchronisation privilégiant l'aspect prioritaire de la ressource partagée.
  - Structure à 5 champs :
    - Une priorité : *PRIOR*
    - Une file d'attente : *F*
    - Un élément FRONT contenant une tâche ayant exécuté un *WAIT*
    - Un champ *OLD-PRIOR*
    - Une variable *STATE* pouvant prendre les états :
      - 1 : Personne n'utilise la Control-Queue
      - 2 : La Control-Queue est utilisée
      - 3 : Qq'un attends dans le FRONT
      - 4 : Un signal à été émis alors qu'on était dans l'état 2
      - 5 : Un signal à été émis alors qu'on était dans l'état 1.
    - 4 primitives associées
      - *JOIN* : correspond au Prendre d'un sémaphore booléen,
      - *LEAVE*: correspond au Vendre d'un sémaphore booléen,
      - *WAIT*: entre *JOIN* et *LEAVE*, passante si *STATE* = 4 ou 5,
      - *STIM*: Si *STATE* = 3 : tâche réactivée, sinon *STATE* = 4 ou 5.

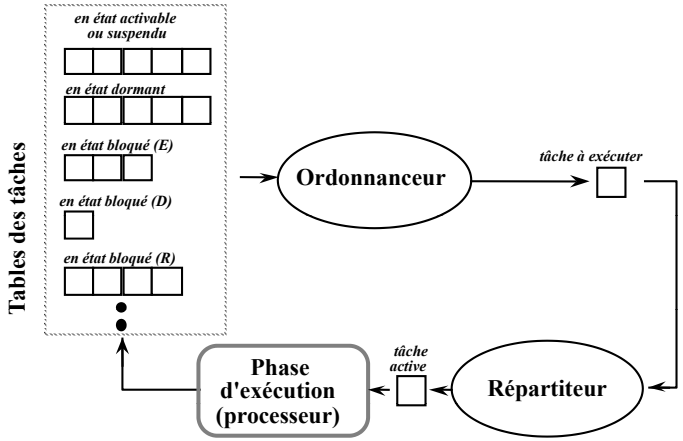
- L'ordonnancement :
  - Principe : planifie l'exécution des tâches, en fonction de l'état de l'environnement, de telle sorte que les échéances temporelles soient respectées
  - Mécanisme : à chaque instant où l'ordonnanceur est activé
    - examine tous les événements arrivés et l'état du système
    - exécute l'algorithme d'ordonnancement mettant en œuvre la stratégie de partage des ressources
    - retire les processeurs aux tâches en activité qui ne sont plus prioritaires
    - sauvegarde leur contexte
    - (re)lance les tâches prioritaires
  - => Questions :
    - quels algorithmes d'ordonnancement ?
    - à quel moment ordonnancer ?

- L'ordonnancement :
  - => Questions : quels algorithmes d'ordonnancement ?
    - problème complexe sans solution dans le cas général (NP complet)
    - => hypothèses simplificatrices :
      - monoprocesseur
      - tâches indépendantes
      - le temps pris par l'ordonnanceur lui-même est négligé
    - => Plusieurs algorithmes :
    - ordonnancement cyclique (les tâches sont exécutées dans un ordre défini à l'avance - pas de préemption)
    - ordonnancement par priorité (le problème revient alors à choisir les priorités)
    - "rate monotonic" : ordo par priorité où la priorité est inversement proportionnelle à la périodicité des tâches
      - optimal
      - correction mathématiquement prouvée
      - mais ne s'applique qu'aux tâches périodiques et indépendantes
    - ordonnancement par échéances…

- L'ordonnancement :
  - => Questions : à quel moment ordonnancer ?
    - si trop fréquent : risque de surcharge du processeur
    - si peu fréquent : risque de manque de réactivité par rapport à l'environnement
    - => Plusieurs stratégies
    - stratégie préemptive : activation de l'ordonnanceur chaque fois qu'une action est susceptible de faire évoluer l'état du système
      - => bonne réactivité mais risque d'augmenter l'overhead (temps pris par le système pour se gérer lui-même)
    - stratégie préemptive cyclique : activation de l'ordonnanceur lors d'instants prédéfinis (par exemple à chaque top d'horloge…)
    - stratégie non préemptive : le processeur ne peut être retiré aux tâches en activité ; l'ordonnanceur ne peut être activé que lorsque la tâche en activité rend le processeur

Le module d'ordonnancement, cœur du noyau temps réel, peut être divisé en deux parties:

- l'**ordonnanceur** (*scheduler*), proprement dit, qui est donc chargé d'appliquer l'algorithme d'ordonnancement en gérant les différentes files ou tables des tâches,
- le **répartiteur** (*dispatcher*) qui réalise l'élection effective de la tâche et le changement de contexte associé.



L'ordonnancement : hors-ligne ou en ligne

- **hors-ligne** : construction d 'une séquence d 'exécution complète sur la base des paramètres temporels des tâches en utilisant une modélisation (réseaux de Petri, …) et simulation (animation ou énumération du modèle).
  - surcoût minimum de l'ordonnanceur qui devient un **séquenceur**
  - peu flexible (adaptation à des changements)
- **en ligne** : choix dynamique de la prochaine tâche à exécuter en fonction des paramètres de la tâche en utilisant une modélisation de l'algorithme d'ordonnancement et simulation de l'exécution.
  - surcoût l'ordonnanceur
  - très flexible (adaptation aux changements)

L'ordonnancement : préemptif ou non préemptif

- **non préemptif** : les tâches ne peuvent être interrompues qu'à des endroits spécifiques et à la demande de la tâche elle-même : fin_de_tâche, attente_signal, …
  - temps de réponse important à prévoir
  - pas besoin de mécanismes de partage de ressources critiques
  - programmation simple
- **préemptif** : les tâches peuvent être interrompues à n'importe quel instant et le processeur affecté à une autre tâche.
  - temps de réponse plus court
  - besoin de mécanismes de partage de ressources critiques

- Choix dynamique : critère de priorité
  - A priorité empirique, indépendante des paramètres temporels.
  - A priorité statique ou dynamique, fonction des paramètres temporels.
- Algorithmes d'ordonnancement de base (préemptif)
  - A priorité statique
    - Sur un paramètre temporel statique (P,R)
    - Calculé hors ligne
  - A priorité dynamique
    - Sur un paramètre temporel dynamique (R(t),L(t))
    - Calculée à chaque nouvel événement de réveil

Ordonnancement temps réel versus ordonnancement classique

Ordonnancement classique

- Équité
- Utilisation des ressources
- Temps réponse

Ordonnancement temps réel

- Respect des échéances
- Surcharge supportée
- Certifiable

Notion de priorité

- Modèles de tâches:
  - Rythme d'occurence
    - Tâche périodique Tp
      ex : lecture de capteurs
    - Tâche non-périodique Tap
      ex : alarme
      - **Tâche sporadique**
        - » On peut définir un temps minimal d'apparition entre deux activations successives.
      - **Tâche apériodique**
  - Contraintes de temps
    - Tâches à contraintes strictes
      - Date de fin d'exécution au plus tard "dure« -> respect obligatoire.
    - Tâches à contraintes relatives
      - Date de fin d'exécution au plus tard « lâche« -> respect souhaitable.

- Modèle de tâches: périodique strict

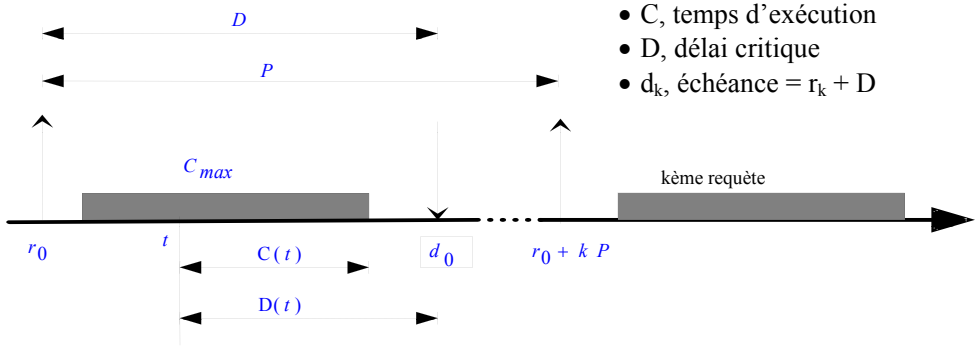
  Tp(r_0, C, D, P)

  0<= C <= D <= P

  Tp(t,C(t),R(t))

  D=P, à échéance sur période: $d_k = r_{k+1}$

- r_0, date de premier réveil
- P période
- r_k, date de réveil de la kème requête
- C, temps d'exécution
- D, délai critique
- d_k, échéance = r_k + D

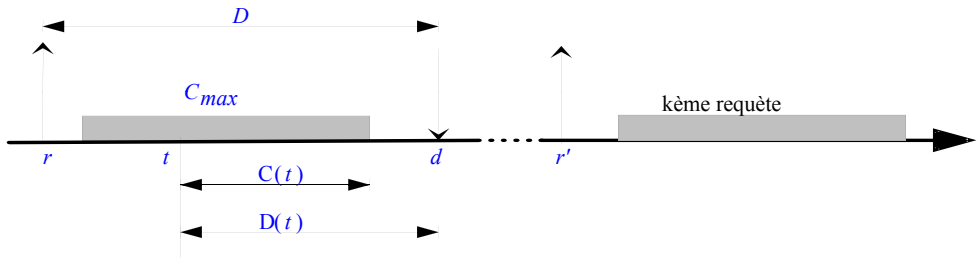


- Modèle de tâches: apériodique strict

  Tap(r, C, D)

  Tap(t,C(t),D(t))

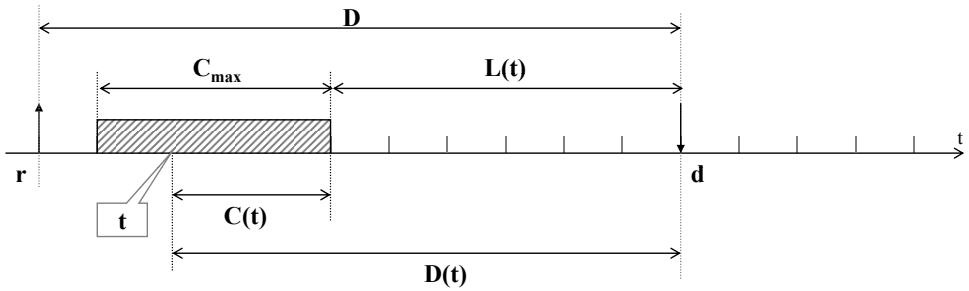
- r, date aléatoire de réveil
- C, temps d'exécution
- D, délai critique
- d_k, échéance = r_k + D



Paramètres dynamiques :

- **Pendant l'exécution de la tâche (à un instant donné t)**
  - **temps d 'exécution restant C(t)** : C(t) = Cmax – Cexécuté
  - **délai critique dynamique D(t)** : temps restant avant la prochaine échéance, **D(t) = d – t**
  - **laxité dynamique L(t)** : temps avant le début d 'exécution de la tâche, **L(t) = D(t) - C(t) = d - t - C(t)**

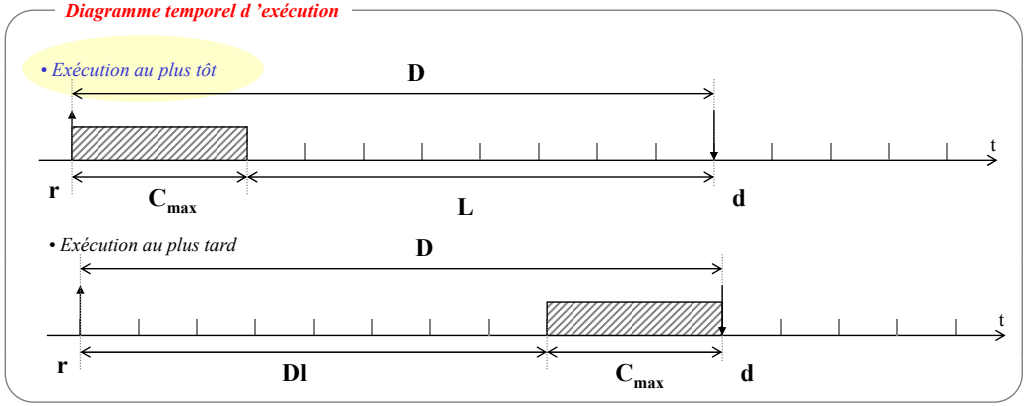
Diagramme temporel d 'exécution



Laxité et Délai de latence :

- **Tâche à contraintes temporelles strictes (périodiques ou apériodiques)**
  - **Laxité** : temps restant entre la fin d 'exécution de la tâche et son délai critique, **Lmax = D – C**
  - **Délai de latence** : temps avant le début d 'exécution de la tâche, **Dlmax = D - C**

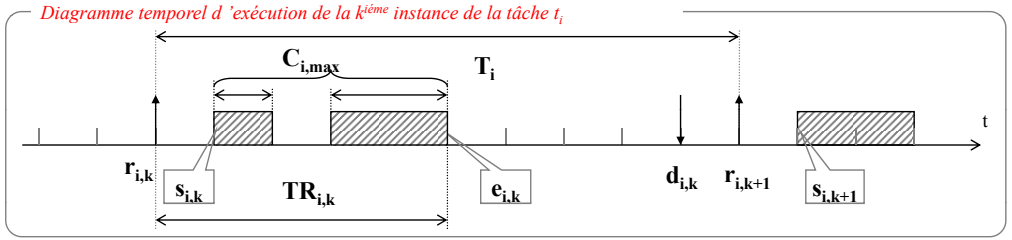
Diagramme temporel d 'exécution



Paramètres d'analyse du comportement d'une tâche

- Début d'exécution de la k^iéme instance de la tâche t_i : $s_{i,k}$
- Fin d'exécution de la k^iéme instance de la tâche t_i : $e_{i,k}$
- **Temps de réponse** de la k^iéme instance de la tâche t_i : $TR_{i,k} = e_{i,k} - r_{i,k}$
  - Temps de réponse maximum de la tâche t_i : $TR_i = max_k\{TR_{i,k}\}$
  - Temps de réponse minimum de la tâche t_i : $TR_{i,min} = min_k\{TR_{i,k}\}$
  - Temps de réponse moyen de la tâche t_i : $TR_{i,moy} = \Sigma_k\{TR_{i,k}\}/k$

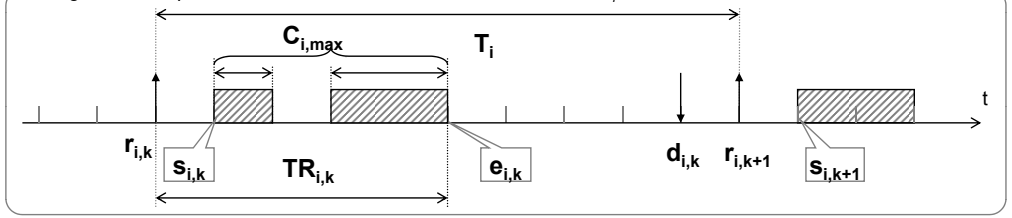
Diagramme temporel d 'exécution de la k^iéme instance de la tâche t_i



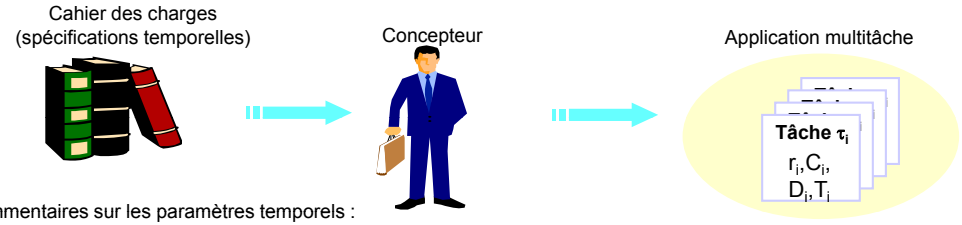
Paramètres d'analyse du comportement d'une tâche

- **Gigue** (régularité d'exécution) entre deux instances consécutives de la tâche τ_i : $g_{i,k} = [(s_{i,k+1} - s_{i,k}) - T_i] / T_i$
  - gigue maximum de la tâche τ_i : $G_i = max_k\{g_{i,k}\}$
  - Temps de réponse minimum de la tâche τ_i : $G_{i,min} = min_k\{g_{i,k}\}$
  - Temps de réponse moyen de la tâche τ_i : $G_{i,moy} = \Sigma_k\{g_{i,k}\}/k$

Diagramme temporel d 'exécution de la k^iéme instance de la tâche τ_i



Intégration des spécifications temporelles dans le modèle de tâches



Commentaires sur les paramètres temporels :

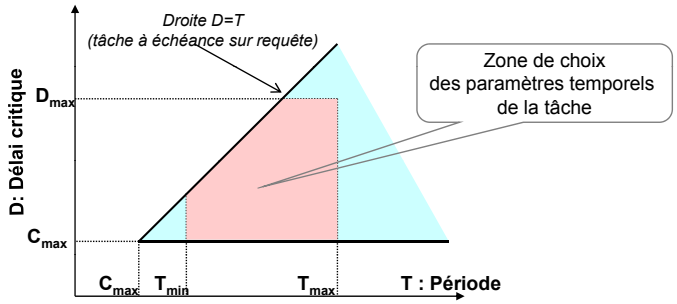
- r_i : Dans le cas général, les tâches sont à départ simultané; mais une tâche peut être retardée au réveil de l'application pour prendre en compte par exemple la précédence. Ce réveil non simultané est obtenu en insérant au début du code de la tâche une primitive DELAI.
- C_i : La durée de la tâche est directement liée au code de la tâche. On évalue généralement la durée d'exécution pire cas C_{max} et une durée minimale C_{min}.
- D_i : Ce paramètre permet au concepteur de limiter le temps de réponse de la tâche. Il peut être codé au niveau d'une tâche en utilisant un temporisateur type "watchdog".
- T_i : Cette périodicité de la tâche est fixée par les besoins de la fonction : tâche de scrutation ou acquisition (polling)

Intégration des spécifications temporelles dans le modèle de tâches

Ainsi viennent s'ajouter des **contraintes temporelles exprimées dans le cahier des charges** de l'application comme par exemple :

- une période maximale T_{max} ($T_i \leq T_{max}$), correspondant, par exemple, à la fréquence minimale d'échantillonnage,
- une période minimale T_{min} ($T_i \geq T_{min}$), correspondant par exemple une période inutile d'acquisition (trop de données à anlyser -redondance),
- une échéance maximale R_{max} ($R_i \leq D_{max}$), afin d'obtenir un meilleur temps de réponse.

→ Certaines tâches ont des paramètres temporels non fixés



➢ Notions dédiées

↳ Notion de fautes temporelles

- Faute temporelle de T = non respect de la contrainte temporelle associée à T.
  - Notion d'**urgence** (choix d'une tâche à exécuter par rapport aux paramètres temporels)

↳ Notion de surcharge

- Occurrence d'une ou plusieurs fautes temporelles.
  - Notion d'i**mportance** (choix d'une tâche à exécuter par rapport aux spécification fonctionnels de l'application)

♦ Analyse d'une configuration de tâches périodiques

♦ **facteur d'utilisation u** : pourcentage du processeur nécessaire à son exécution sur sa période $\mathbf{T_i}$:

$u = C_i / T_i$

♦ **facteur de charge u_l** : pourcentage du processeur nécessaire à son exécution sur son délai critique $\mathbf{D_i}$:

$u_l = C_i / D_i$

♦ **facteur d 'utilisation U** (resp.le facteur de charge U_l) : somme des facteurs d'utilisation $\mathbf{u_i}$ des tâches (resp. des facteurs de charge des tâches $\mathbf{u_{li}}$) de la configuration de **n** tâches périodiques : $T = \{t1, t2, \ldots tn\}$, :

$U = \Sigma_i u_i = \Sigma_i (C_i / T_i)$ $\qquad$ $U_l = \Sigma_i u_{l,i} = \Sigma_i (C_i / D_i)$

Exemple :

Tâche	r_0	C	D	T
1	0	2	6	6
2	2	1	8	8
3	0	2	10	12

$u_1 = 0{,}33$, $u_2 = 0{,}125$, $u_3 = 0{,}166$ **U = 0,625**

$u_{l1} = 0{,}33$, $u_{l2} = 0{,}125$, $u_{l3} = 0{,}2$ $\mathbf{U_l = 0{,}658}$

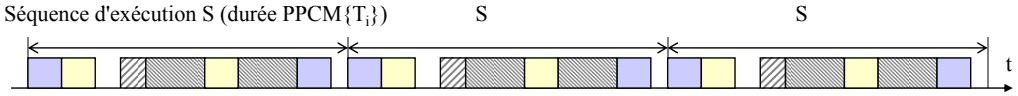
♦ Période d'étude

Pour une configuration de n tâches périodiques : $T = \{t_1, t_2, \ldots t_n\}$, l 'étude de la séquence d 'exécution, produite par un algorithme d 'ordonnancement donné, est limité à un temps **H** appelé période d 'étude ou méta-période ou cycle majeur. Cette période d 'étude est :

Cas où toutes les tâches de l'application sont à départ simultané ($\forall i$, r_i=0) : $H = PPCM\ \{T_i\}$

Exemple :
H = PPCM(6,8,12)=24

Tâche	r_0	C	D	T
1	0	2	6	6
2	0	1	8	8
3	0	2	10	12



♦ Période d'étude

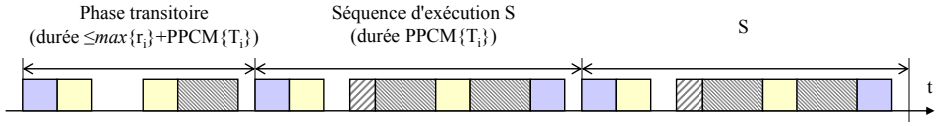
Cas où toutes les tâches de l'application ne sont pas à départ simultané ($\exists(i,j)/\ r_i \neq r_j$) :

$H \leq max\{r_i\} + 2PPCM\ \{T_i\}$

Exemple :

H < 2 + 2*24 = 50

Tâche	r_0	C	D	T
1	0	2	6	6
2	2	1	8	8
3	0	2	10	12



Ordonnancement Temps Réel
3.2. Modèles de tâches

- Notion d'acceptabilité
  - Une Configuration Cfp de tâches Tp périodiques est dite **acceptable ou valide** si toutes les tâches Tp de Cfp respectent les contraintes temporelles qui leur sont associées.
  - **Test d'acceptabilité** associé à O
- Notions d'ordonnançabilité
  - Une Configuration Cfp de tâches Tp périodiques est dite **ordonnançable** si toutes les tâches Tp de Cfp respectent les contraintes temporelles qui leur sont associées **quelles que soient les dates de déclenchement et sur une durée infinie.**
  - L'algorithme qui donne une séquence ordonnançable est dit **fiable**.
- Notion d'optimalité
  - Un algorithme d'ordonnancement O est dit optimal si, étant donnée Cf ordonnançable, le test d'acceptabilité associé à O décrète Cf acceptable

Condition nécessaire d'ordonnançabilité

$$U = \sum_i \frac{C_i}{T_i} \leq 1$$

Ordonnancement Temps Réel
3.2. Modèles de tâches

Algorithme d'ordonnancement
(affectation de priorités)
+
Configuration de tâches (r_i,C_i,D_i,T_i)
(**U** et $\mathbf{U_l}$)

Optimalité ?

Optimalité de l'algorithme d'ordonnancement dans ce contexte

Ordonnançabilité ?

Condition suffisante ou nécessaire et suffisante d'ordonnançabilité de la configuration

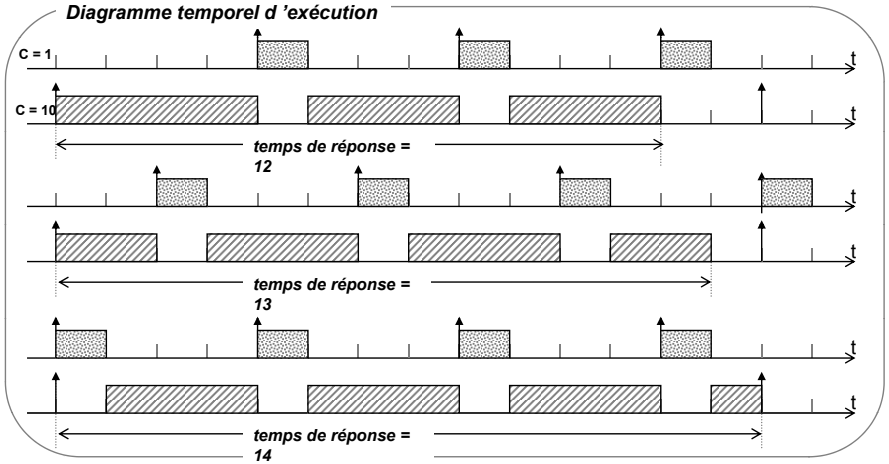
Simulation de l'exécution afin de construire tout ou partie de la séquence d'exécution

Tâches indépendantes (périodiques ou apériodiques) → **Tâches dépendantes** → **Tâches avec partage de ressources critiques**

Ordonnancement Temps Réel
3.2. Modèles de tâches

♦ **Instant Critique :**

➢L'ordonnançabilité d'une configuration peut être vérifiée lorsque les tâches sont toutes à départ simultané : activation au même instant appelé **instant critique** (pire cas).



Ordonnancement Temps Réel
3.3. Tâches Indépendantes

- Algorithme Rate Monotonic (Liu & Layland 73)
  - Critères d'application:
    - Les tâches sont périodiques et à l'état PRÊT.
    - Les tâches peuvent être préemptées mais on néglige le temps de commutation et d'ordonnancement.
    - Le temps d'exécution C_{max} est connu.
  - Priorité statique basée sur la période (HPF)

$$\text{Prio}(T_i) = \frac{1}{P_i}$$

    - La tâche la plus prioritaire est celle de plus petite période.
  - Optimal pour la classe des algorithmes à priorité statique pour des configurations Cfp de n tâches à échéances sur requête
    - Test d'acceptabilité (Condition Suffisante).

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq n\left(2^{\left(\frac{1}{n}\right)} - 1\right)$$

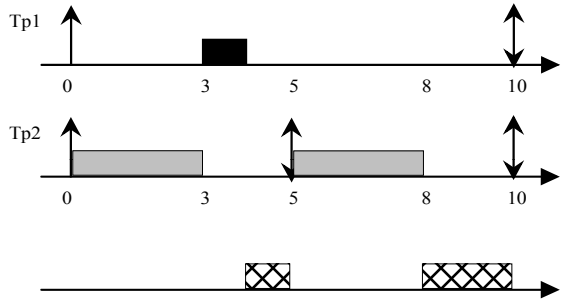
Algorithme Rate Monotonic (High Prio First)

- Exemple :

  Tp1(r_0=0, C = 1, T = 10)
  Tp2(r_0=0, C = 3, T = 5)

  Prio(Tp1) =1/10 < Prio (Tp2) = 1/5

  Test d'acceptabilité : 1/10 + 3/5 = 0,7 < 0,82



Théorème de la zone critique *(Rajkumar, 91)*

- La condition du test d'acceptabilité de R.M. est très restrictive.
- Si toutes les tâches arrivent initialement et respectent leur première échéance, alors toutes les échéances seront respectées par la suite.

- Test de terminaison

  Un ensemble de **n** tâches T {t1, t2, t3, …, ti, …, tn} ordonnées suivant les priorités (**t1** la tâche la plus prioritaire et **tn** la tâche la moins prioritaire) définies par les paramètres temporels (**ri, Ci, Di, Ti**) est ordonnançable si et seulement si :

$$\forall i,\ 1 \le i \le n \quad \min_{0 \le t \le D_i} \sum_{j=1}^{i} \frac{C_j}{t} \lceil t/T_j \rceil \le 1$$

avec ⌈x⌉ valeur entière immédiatement supérieure ou égale à x

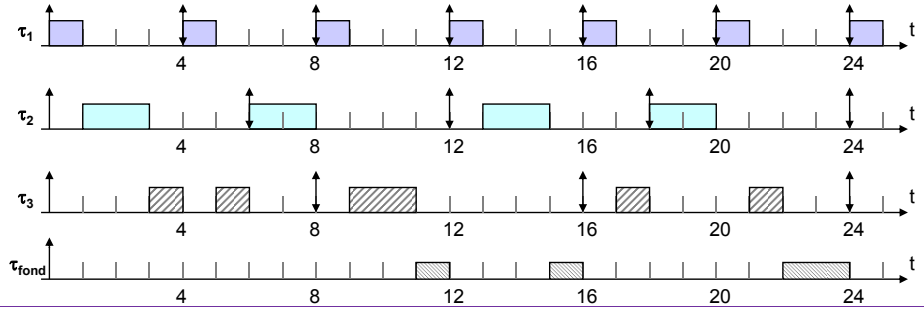
- Méthode itérative : on cherche t tq W(t) = t

$$W(t) = \sum_{j=1}^{i-1} C_j \lceil t/T_j \rceil + C_i = t \quad \le D_i \quad ?$$

Théorème de la zone critique : exemple

T	r_i	C_i	D_i	T_i	u_i	$Prio_i$
τ_1	0	1	4	4	0,25	3
τ_2	0	2	6	6	0,33	2
τ_3	0	2	8	8	0,25	1

- Facteur d'utilisation **U = 0,83 (> 0,78)**
- Période d'étude **H = 24**
- Temps libre processeur T_{libre} **= 4**



Théorème de la zone critique : preuve de l'ordonnançabilité de l'exemple

On utilise la relation permettant de prouver l'ordonnançabilité de cette configuration sans simulation à l'aide du théorème de la zone critique. Le calcul est effectué jusqu'au temps D_i à chaque nouvelle activation de tâche :

☆ i = 1 : $\frac{C_1}{t}\left\lceil \frac{t}{T_1} \right\rceil$ (D_1=4) • pour t = 4 ⟹ $\le D_1$

i = 2 : $\frac{C_1}{t}\left\lceil \frac{t}{T_1} \right\rceil + \frac{C_2}{t}\left\lceil \frac{t}{T_2} \right\rceil$ • pour t = 4

(D_2=6) • pour t = 6

min ⟹ ≤ 1

i = 3 : $\frac{C_1}{t}\left\lceil \frac{t}{T_1} \right\rceil + \frac{C_2}{t}\left\lceil \frac{t}{T_2} \right\rceil + \frac{C_3}{t}\left\lceil \frac{t}{T_3} \right\rceil$ • pour t = 4

(D_3=8) • pour t = 6

• pour t = 8

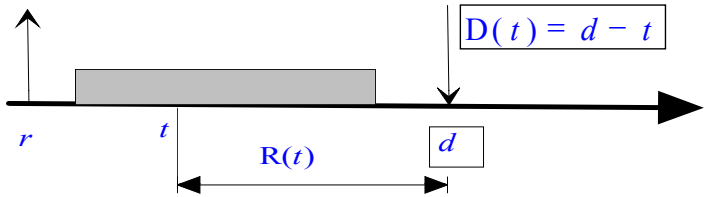
min ⟹ ≤ 1

Algorithme Earliest Deadline First (EDF)

- Priorité Dynamique basée sur le délai critique dynamique

$$\text{Prio}(T_i)(t) = \frac{1}{D_i(t)}$$

La tâche la plus prioritaire est celle dont l'échéance est la plus proche



Algorithme Earliest Deadline First (EDF)

- Optimal pour la classe des algorithmes à priorité dynamique pour des configurations Cp de **tâches quelconques.**
  - CNS d'acceptation pour Cp de tâches à échéance sur requête

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

  - CS pour Cp de tâches quelconques.

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

  - CN pour Cp de tâches quelconques

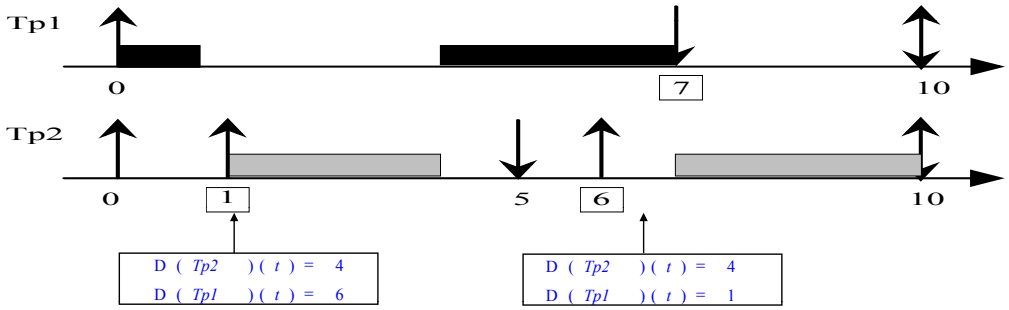
$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

Algorithme Earliest Deadline First (EDF)

- Exemple :

Tp1 $(r_0 = 0, C = 4, D = 7, P = 10)$
Tp2 $(r_0 = 1, C = 3, D = 4, P = 5)$

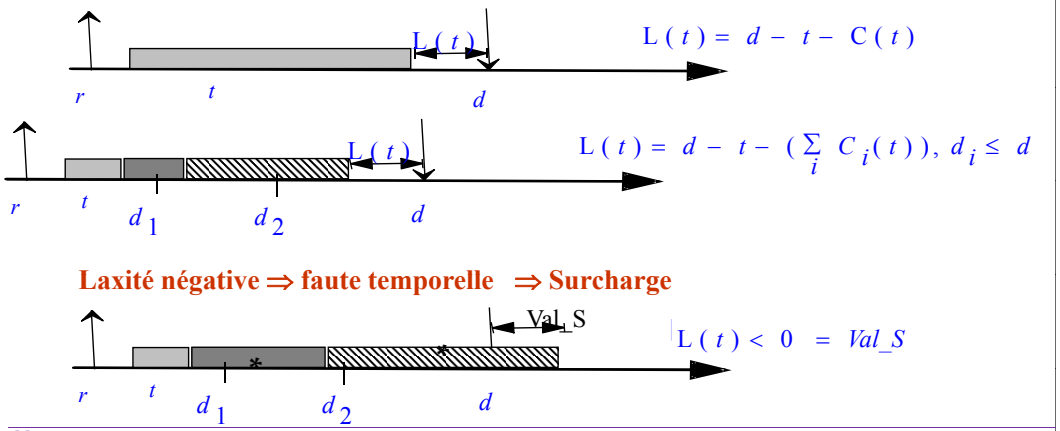
CS : 4/7+3/4 = 1,32
CN : 4/10+3/5 = 1



Algorithme Least Laxity First (LLF)

- Marge la plus courte d'abord
- Critère d'acceptabilité identique à EDF

Exemple de Laxité dynamique à t



Ordonnancement avec contraintes de ressources

- Mêmes hypothèses que dans le cas « Rate Monotonic »
- Ordonnancement en présence de sections critiques :
  - Gestion de l'inversion de priorité et des interblocages

Problème de l'inversion de priorité

- Une tâche de plus faible priorité bloque une autre tâche de plus forte priorité quand elle est en S.C.

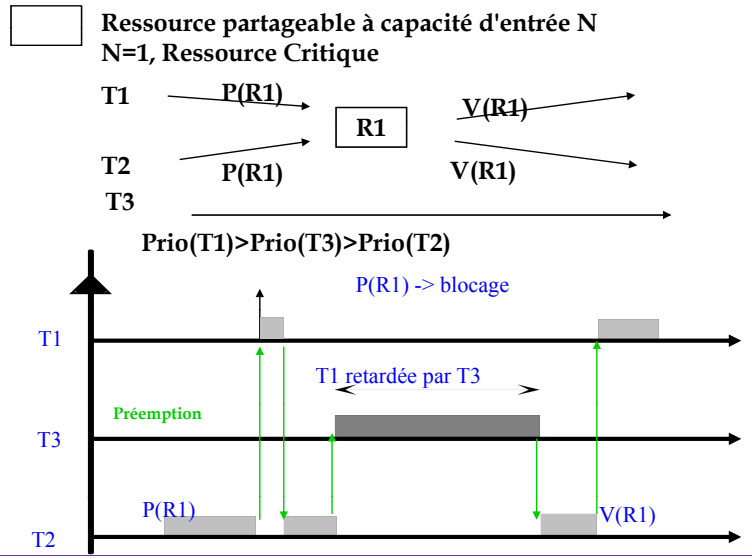
↳ Pouvoir borner l'attente d'une tâche prioritaire et inclure cette borne au test d'acceptabilité de la configuration à ordonnancer

- Test d'acceptabilité de Cfp :
  - Test classique + facteur de blocage B

↳ Eviter les situations d'interblocage

- Protocole de l'héritage de priorité [SHA 90]
- Protocole de la priorité plafond [SHA 90]

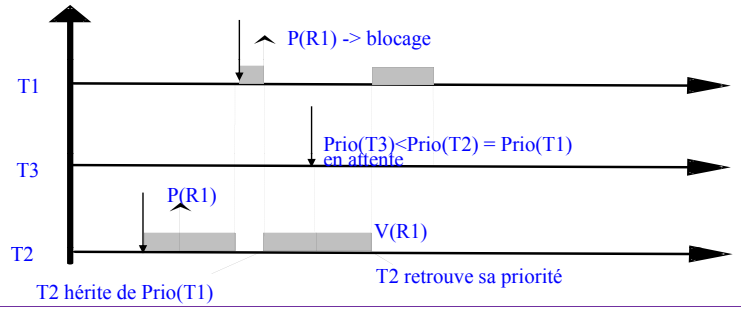
Exemple de problème de l'inversion de priorité



Problème de l'inversion de priorité

↳ Protocole de l'Héritage de Priorité

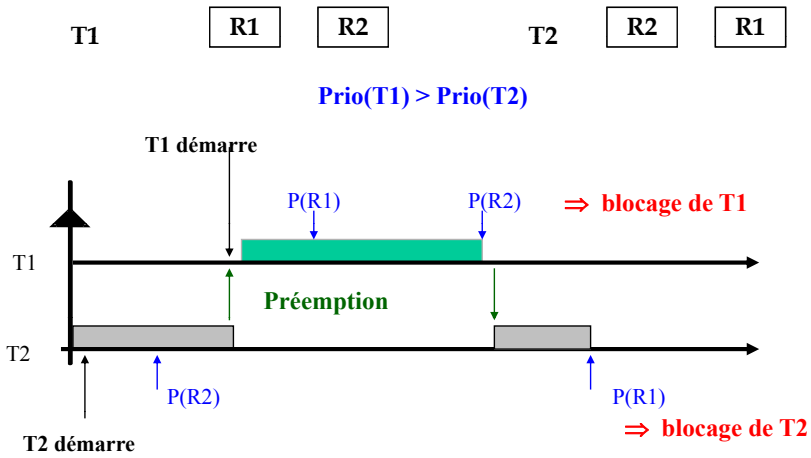
- Une tâche en section critique hérite de la priorité de la plus haute tâche en attente sur la section critique.
- Section critique terminée au plus tôt, T1 retardée au plus de la somme des durées des sections critiques partagées avec des tâches de plus faible priorité (ici T2).
- **MAIS ⇒ NE prévient pas les Interblocages.**



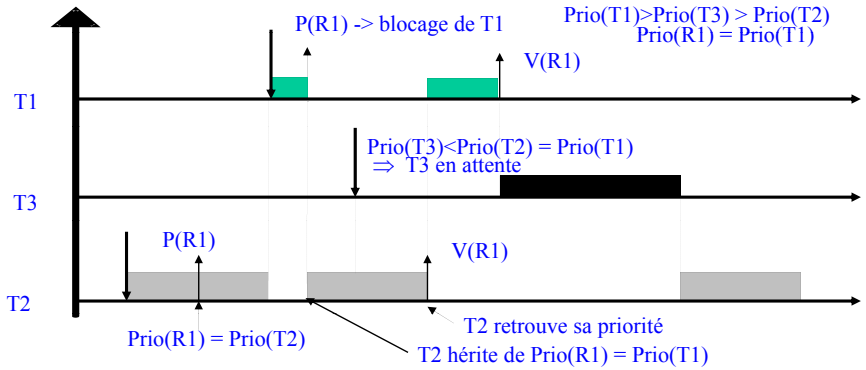
Problème de l'inversion de priorité

↳ Protocole de l'Héritage de Priorité

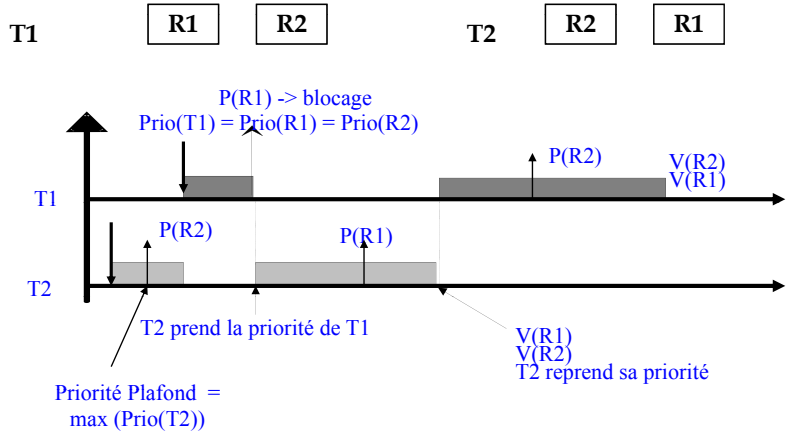
- Problème de **l 'Interblocage**



- Problème de l'inversion de priorité
  - Protocole de la Priorité Plafond
    - Une section critique reçoit une priorité telle que
      - Prio(S) = max(Prio(T_i), T_i accède à S)
    - Une tâche possédant S prend la priorité de S.
    - Une tâche ne peut prendre S que si sa priorité est supérieure à toutes les priorités des sémaphores S' détenus par les autres tâches.



- Problème de l'inversion de priorité
  - Protocole de la Priorité Plafond
    - plus de problème d'**Interblocage**

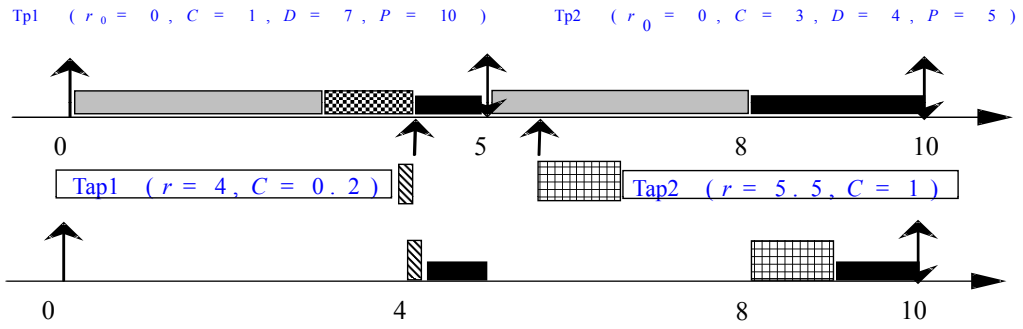


- Problème de l'inversion de priorité

Protocole	Ordonnancement	Propriétés
Héritage de priorité	**RM**	• **Test d'ordonnançabilité** • **Pas de prévention des interblocages**
Priorité plafonnée	**RM**	• **Test d'ordonnançabilité** • **Prévention des interblocages**
Priorité plafonnée dynamique	**ED**	

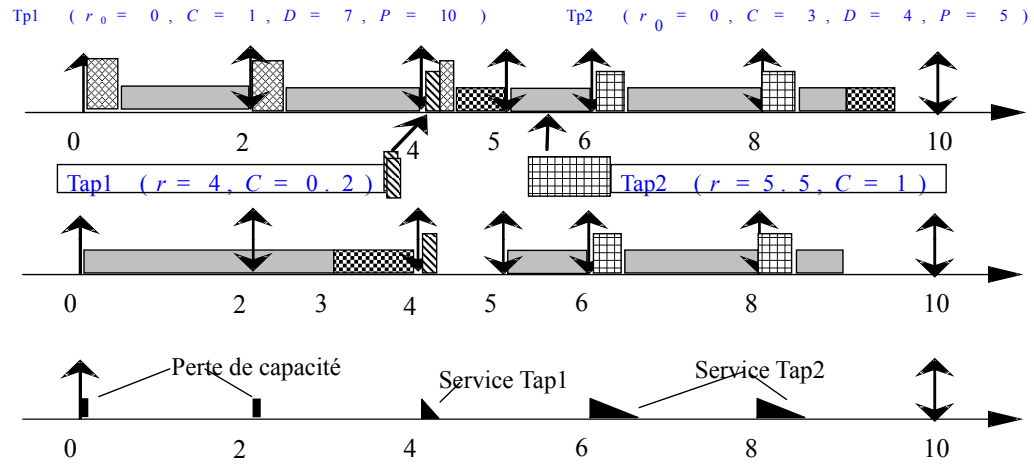
- Ordonnancements spécifiques
  - Dépend des spécifications du problème
  - Permet de traiter la surcharge
- Critères de classification
  - Algorithmes des tâches périodiques
  - Criticité des tâches apériodiques
  - Niveau de prise en compte des tâches apériodiques / tâches périodiques
- Solutions :
  - Deux grandes orientations :
    - Taches de fond
    - Taches prioritaires : Notion de serveur de Taches apériodiques
      - Serveur de Scrutation (Polling Server)
      - Serveur Différé (Deferrable Server)
      - Serveur Sporadique (Sporadic Server)

- Premier groupe : Traitement d'arrière plan :
  - Tâches périodiques : RM
  - Tâches apériodiques à contraintes relatives
  - Les tâches apériodiques sont ordonnancées selon un ordre FIFO lorsque le processeur est oisif, dans les temps creux de la configuration périodique
    - Temps de réponse / charge périodique



- Deuxième groupe : Tâche de plus haute priorité :
  - Tâches périodiques : RM ou EDF
  - Tâches apériodiques à contraintes fortes (ou non connues)
  - Minimiser les temps de réponse
    - Notion de Serveurs
      - Tâche périodique dédiée de plus haute priorité : Tps (r_0, C, P), tâche de Cfp
      - Serveur de Scrutation, Différé et Sporadique
    - Serveur divisé en deux parties :
      - L'une regarde s'il y a des taches apériodiques à traiter,
      - L'autre sert à traiter la tâche apériodique.
    - Le serveur sert les tâches arrivées dans l'intervalle
      - [n-1 T_{poll} ; n T_{poll} [
    - Quand un événement arrive, il est pris en compte avec une capacité de traitement Cmax
    - Si le temps de traitement de la tâche apériodique dépasse Cmax, elle est interrompue jusqu'a la prochaine activation du serveur

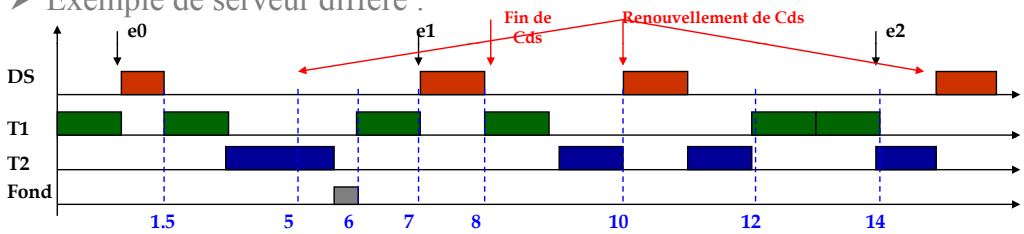
- Serveur de scrutation : Tpsc (r_0 = 0 , C = 0 , 5 , P = 2)



- Notion de serveurs apériodiques
  - Inadéquation entre le rythme périodique du serveur de scrutation et le caractère aléatoire des réveils apériodiques
    - Modifier la gestion de la capacité du serveur tel que le serveur dispose au plus vite de capacité pour servir une nouvelle tâche périodique
    - Serveur Différé (Cds, Tds)
      - Il ne consomme pas de temps CPU tant qu'il n'y a pas de tâche apériodique à exécuter.
      - Si la capacité du serveur Cds n'est pas totalement utilisée, elle est perdue.
      - Si la capacité du serveur Cds ne suffit pas pour traiter la tâche apériodique, le traitement continue lors du renouvellement du serveur (Activation à Tds).
      - Généralement, on assigne la plus haute priorité au serveur (Tds < Min(tâches périodiques)).
    - Serveur Sporadique (Css, Tss)
      - La période de renouvellement n'est pas fixe, mais dépend de l'instant d'activation du serveur et de sa période de réapprovisionnement (Tss).
      - La capacité du serveur Css est consommée au fur et à mesure de l'arrivée des événements à servir.

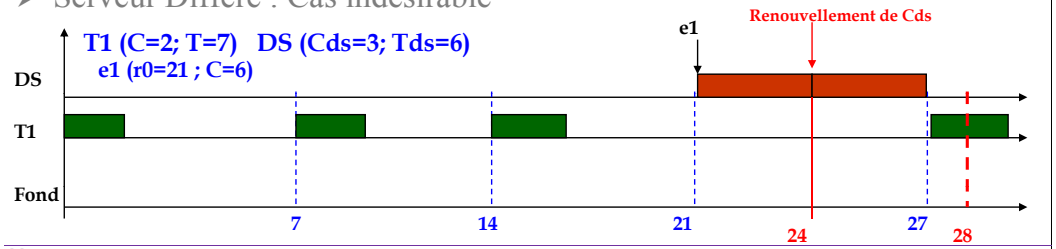
Ordonnancement Temps Réel
3.5. Prise en compte de tâches apériodiques

➢ Exemple de serveur différé :



T1 (C=2; T=6) T2 (C=3; T=8) DS (Cds=1; Tds=5)
e0 (r0=1 ; C=0.5) e1 (r0=7 ; C=2) e2 (r0=14 ; C=1)

➢ Serveur Différé : Cas indésirable



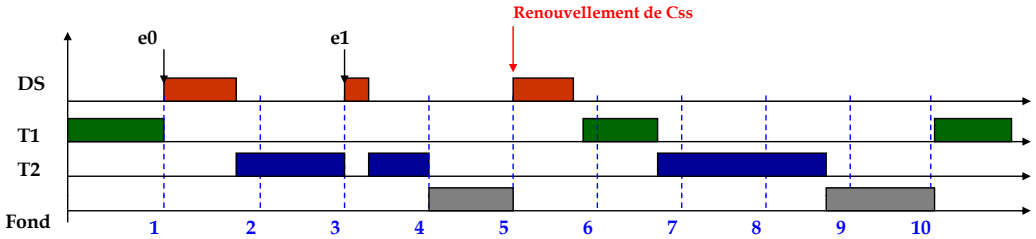
Ordonnancement Temps Réel
3.5. Prise en compte de tâches apériodiques

➢ Exemple de Serveur Sporadique :

SS (Css=1; Tss=4)

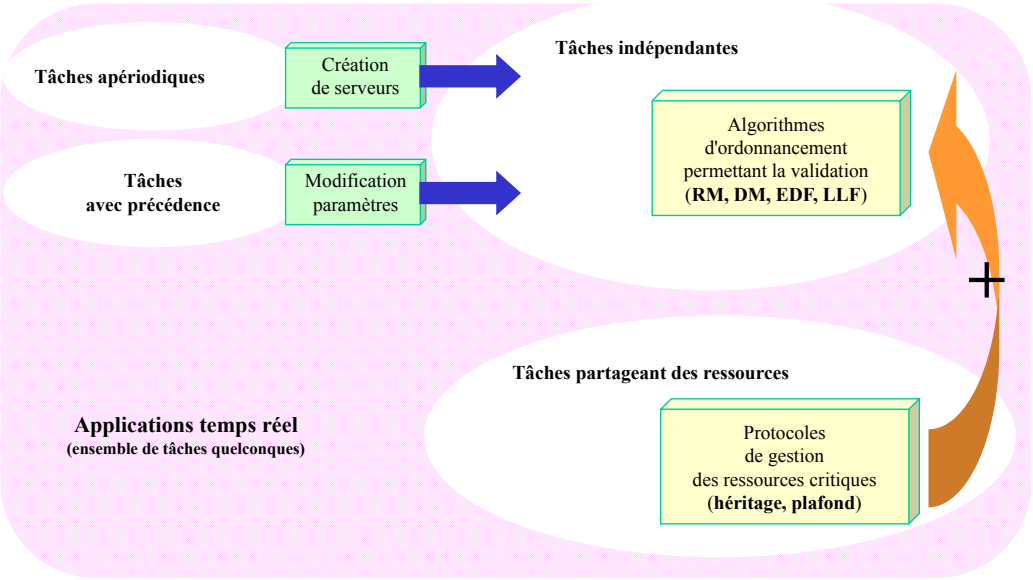
T1 (C=1; T=5) T2 (C=2; T=6)

e0 (r0=1 ; C=0.75)
e1 (r0=3 ; C=1)



Instant de renouvellement : t=5 ⇐ Activation à 1 + Tss

Ordonnancement Temps Réel
3.6. Conclusions



Ordonnancement Temps Réel
3.6. Conclusions

Rate Monotonic	Earliest Deadline First	Least Laxity First
• Priorité fixe • 1 / Période	• Priorité dynamique • 1 / Délai critique dynamique	• Priorité dynamique • 1 / Marge dynamique
• Test d'ordonnançabilité Charge totale sous la contrainte : $n(2^{1/n} - 1)$ • Modèle de tâche restreint Périodique à échéance sur requête	• Test d'ordonnançabilité Charge totale < 100% • Délai critique ⇨Modèle de tâches plus large tâches Tp et Tap	• Test d'ordonnançabilité Charge totale < 100% • Possibilité de détecter le non respect des contraintes de temps
• Le plus prisé • On peut prendre en compte les temps de préemption ou de commutation de contexte dans les capacités Ci	• Meilleur algo quand la loi d'arrivée des tâches est quelconque et que l'on ne connaît pas leur capacité • Optimal (tout ordonnanc. arbitraire peut être transformé en EDF • Non stable, toutes les tâches doivent respecter leur échéances. • Entraîne le moins de commutations de contexte	• Meilleur algo quand la loi d'arrivée des tâches est quelconque et que l'on connaît leur capacité • Peut prendre en compte les situations de blocages

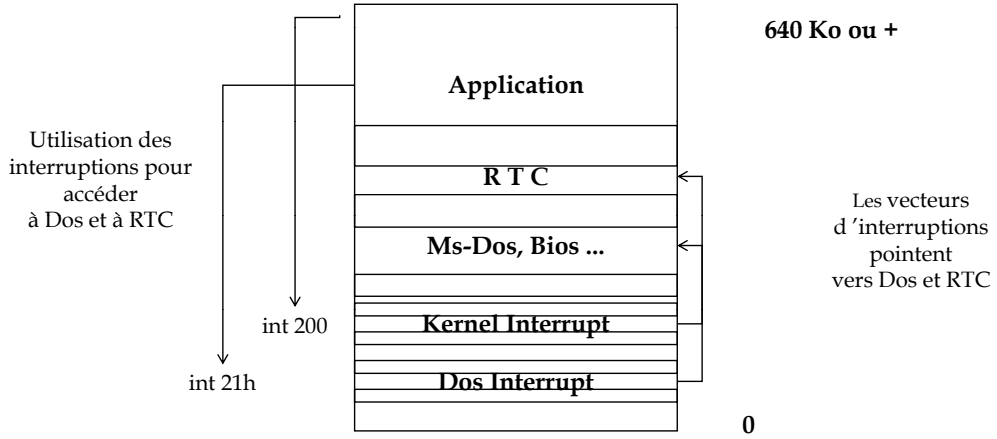
Extensions aux protocoles de réseaux IEEE 802.4 et IEEE 802.6
⇒ RM et EDF

Exemple de noyau Temps Réel
4.1. RealTime Craft

- Type « Noyau T.R. » conçu en France par la société GSI - Tecsi
  - Noyau très proche du Projet SCEPTRE
  - SNECMA : pilotage de moteur d'avion (rafale)
  - Projet « Prometheus » : l'automobile du Futur
- Caractéristiques générales :
  - 27 fonctions (1 vecteur d'interruption par fn)
  - 4 états pour une tâche
  - Temps de commutation de contexte : 8 µs
  - Temps de réponse à une interruption : 15 µs
  - 6 niveaux de priorité (0-5)
    - Ordonnancement HPF
    - 1 file par niveau de priorité
    - Niveau 0 : tâche de fond
  - Notion de composant logiciel : Livré en EPROM 3 Ko

Exemple de noyau Temps Réel
4.1. RealTime Craft

- Fonctionnement sur PC :
  - Dimensionnement, initialisation par appel de primitive `InitKernel`



Exemple de noyau Temps Réel
4.1. RealTime Craft

- Description d'une tâche
  - Code de la tâche : code d'une fonction
  - Ses données
  - Pile
  - Le Contexte d'une Tâche (TCB)
    - Pointeur de chaînage de tâches
    - Pointeur vers Ready List (priorité)
    - Evénements
    - Message transmis par Send
      - Attendus
      - Reçus
    - Etat
    - Temps restant, temporisation
    - Pointeur vers sémaphore ou boîte aux lettres
- Noyau gère les tables
  - Des tâches
  - Des autres « objets »

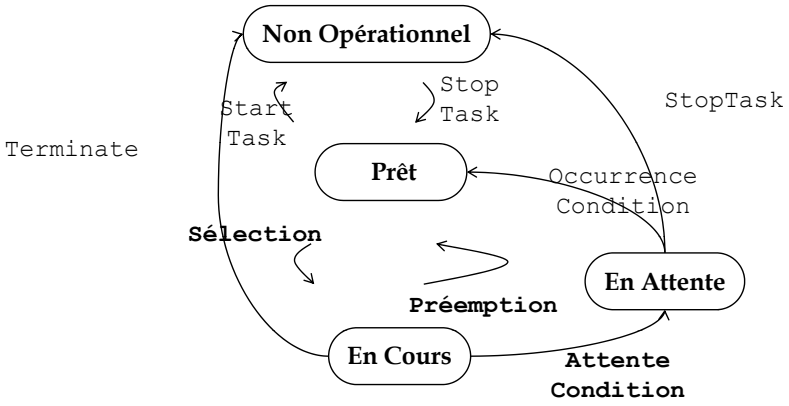
Exemple de noyau Temps Réel
4.1. RealTime Craft

- Gestion des tâches
  - `InitTask` Donne les élts de description
  - `StartTask` Rend la tâche prête
  - `StopTask` Suspend la tâche
  - `Terminate` Auto arrêt
  - `CurrentTask` Permet de connaître son id
  - `TaskState` Connaître état d'1 tâche
  - `TaskPriority` Connaître priorité d'1 tâche
  - `ChangePriority` Changer priorité d'1 tâche
  - `EnterRegion` Disable Task Rescheduling
  - `LeaveRegion` Enable Task Rescheduling
- « Démarrage » :

```
err = StartTask (TaskNumber);
```

  - Code retour
    - cOK : opération réussie
    - cBadTaskNumber : <TaskNumber> n'est pas compris entre 0 et ConfigTable.NbTask-1
    - cTaskAlreadyStarted : la tâche <TaskNumber> n'est pas dans l'état non opérationnel
    - cTaskNotInitialised : La tâche n'a jamais été initialisée depuis InitKernel

- Les différents Etats d'une Tâche :



- Exclusion Mutuelle
  - Communication et partage de ressources
- Hiérarchie de priorité
  - Interruptions - ordonnées
    - Noyau
    - Gestionnaires d'E/S
    - Tâches - ordonnées
- Méthodes de plus en plus « sophistiquées »
  - Masquage des interruptions
    - Sauf sur certains systèmes récents
  - Bloquage de l'ordonnanceur

    `EnterRegion();{accès ressource} LeaveRegion();`
  - Explicite par un sémaphore
    - Définit un jeton d'accès
    - Inconvénients
      - Risque de verrouillage de la ressource
      - Violation de priorité
      - Verrou mortel (deadlock)

- Synchronisation
  - Sans mémoire
    - Drapeaux (flags) d'événements (events)
      - Lié à une tâche
    - Ensemble : attente OU
    - Traduction directe d'une transformation de contrôle
    - StopTask/StartTask :
      - Pour une tâche spécifique
    - Permet de gérer un graphe d 'état
  - Avec mémoire
    - Sémaphores
    - Une seule tâche libérée à la fois

- Gestion des Sémaphores
  - Définition / destruction
    - Nombre défini à l'initialisation du noyau
    - Repéré par son index
    - Définition du nombre max. de jetons par :
      - ce n'est pas une initialisation du nombre courant

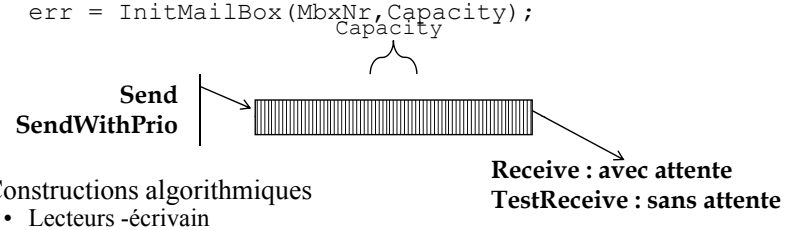
    `err = InitSemaphore( SemNumber, CountMaxValue);`
  - Demande de jeton : P(S)
    - Bloquage si pas de jeton libre

    `err = P(SemNr, timeout);`
    `err = PWithPrio(SemNr, timeout);`
  - Libération d 'un jeton : V(S)
    - Libère une tâche en attente : `err = P(SemNr);`
  - Décrément sans attente si jeton disponible

    `err = TestP(SemNr);`
- Utilisation
  - Exclusion mutuelle
  - Synchronisation / Rendez-vous
  - Comptage de retard, chien de garde
  - …

- Primitives de communication
  - Primitives système : Boîtes aux lettres
    - Passer des pointeurs pour communiquer des structures
    - Initialisation
    - Nombre de boîtes aux lettres + nombre d'enveloppes définis à l'initialisation du noyau
    - Repéré par son Index
    - définition du nombre de message max..

```
err = InitMailBox(MbxNr,Capacity);
```



  - Constructions algorithmiques
    - Lecteurs -écrivain
    - Producteur consommateur
    - Client-Serveur (communication et partage)
    - …

- Gestion du Temps
  - 1 tâche horloge
    - Priorité définie à l'initialisation
    - Recherche/ mise à jour des timeouts
    - Routines à délai
    - Si pas d 'occurrence d 'IT horloge entre temps
      - appelées à l'échéance de leur délai
      - Désactivation de la tâche horloge
    - sinon nouvelle exécution
  - Activée sur IT horloge
    - Horloge PC à 18.2 Hz

```
RTCClockInit( XecTicks);
RTCClockReset();
```

    - Horloge carte PCMES à la fréquence du procédé
      - voir code joint.

- Gestion du Temps
  - Ajout/suppression de routines à délai

```
typedef Word (* tfpFunc) ();
Byte DelayLnk[25];
extern "C" far ConnectDelay ( Byte *, Word, void (*)(void)
  );
ConnectDelay(DelayLnk,DataSegment(),routine);
err = StartDelay(DelayNr,Firstd,Period,DelayLnk);
err = StopDelay(DelayNr);
```


- Utilisation des primitives du DOS
  - pbe de réentrance
    - Gestion fichier
    - Affichage
    - ...
  - Attention gestion mémoire Dos (malloc,new) incompatible
    - Résultats incohérents
    - Durée non déterministe
  - Exclusion mutuelle des appels DOS
    - A la main
    - Avec RTC

```
InitProtect (Itnoyau,ITs, Sémaphores);
ResetProtect();
```

- Initialisation

```
InitKernel(
   &ConfigTable,
   &Bkg_TCBAddr,
   &Clk_TCBAddr
);
```

Exemple de noyau Temps Réel
4.1. RealTime Craft

- Conclusions :
  - Noyau
    - Dimensionnement au plus près de l 'espace mémoire
    - Utilisation seul ou derrière Ms-Dos
  - Tâches
    - Modèle de parallélisme dans un système informatique
  - Gestion dynamique de la mémoire
    - Inexistante en standard
  - Synchro communication
    - 3 objets systèmes
      - Groupes d 'événements
      - Sémaphores
      - Boîtes aux lettres
    - Schémas complémentaires
  - Maîtrise du temps
    - Temps d 'appel des primitives constant quelle que soit la charge
    - Existence de timeout ou de demandes de services non bloquants
    - Routines périodiques toutes au même niveau

Exemple de noyau Temps Réel
4.2. VxWoks

- VxWorks : OS commercial temps réel développé par Wind River Systems.
- Idée principale:
  - use monolithic kernel to schedule user tasks according to user defined priorities. Maximize kernel timing predictability.
  - Gives the users maximal control.
  - A dedicated real time system, not intended as a general purpose OS.
- Caractéristiques:
  - Lacks many modern os features that interfere with real time performance (flat memory model, no paging).
  - Scheduling is done using a preemptive priority driven approach, priorities are chosen arbitrarily by the user (0-255).
  - Priorities can be changed by the user at runtime but this is discouraged.
  - A user can lock a task so that it can't be preempted even by higher priority tasks or interrupts.
  - This allows the use of the fixed priority response time analysis to check schedulability offline.
  - Is resource sharing aware and has a priority inheritance built in.
  - Optimizations in implementation of the context switch and the return from interrupts.
  - The kernel never disables NMI (non-maskable interrupts) so they are always available to the user.

Exemple de noyau Temps Réel
4.2. VxWoks

- LIMITATIONS:
  - Ne propose pas les primitives d'un OS "moderne".
  - Le respect des échéances est du ressort du concepteur informatique.
  - Ne supporte pas les API "modernes" (petite partie de POSIX).
  - Le modèle de mémoire retenu est source de fragmentation mémoire.
  - Offre un maximum de contrôle, mais peu de services
- OS dédié aux applications Temps Réel.

Méthode de modélisation
5.1. SART

- Objectifs :
  - Décrire le cycle de vie du produit
    - De la Spécification à l'Installation / Maintenance
  - Tenir compte des contraintes T.R.
    - 4ème rang dans le Diagramme de BOEHM (paramètres affectant le coût d'un logiciel)
  - Impératif de coût, qualité, fiabilité, portabilité, réutilisabilité, …
- Contraintes :
  - Evolution temporelle des composants du système
  - Interaction système / environnement
    - Difficulté de l'intégration complète
      - Pas de STANDARD en TR
- Méthodes d 'analyse et de conception :
  - Basées sur SA de Yourdon et DeMarco (1979)
    - Approche informationnelle, Fonctionnelle et temporelle.
  - Evolution vers SA / RT
    - Méthode de WARD & MELLOR (1985)
    - Méthode de HATLEY & PIRBHAI (1987).

Méthode de modélisation
5.1. SART

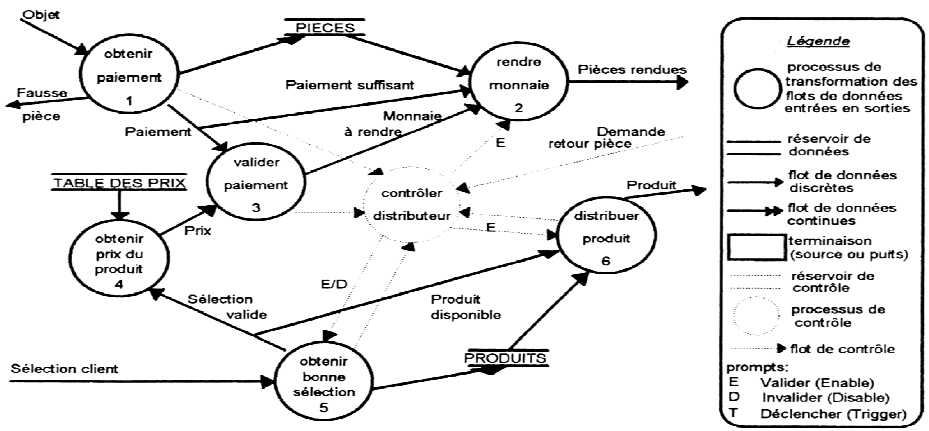
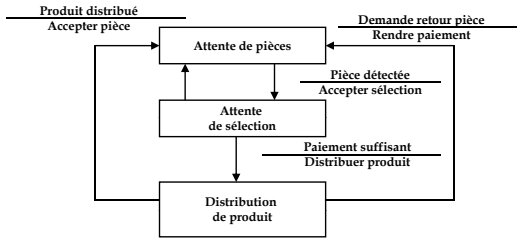


Figure 3.2. *DFD de SART (Ward & Mellor) d'un distributeur de boisson*

Méthode de modélisation
5.1. SART

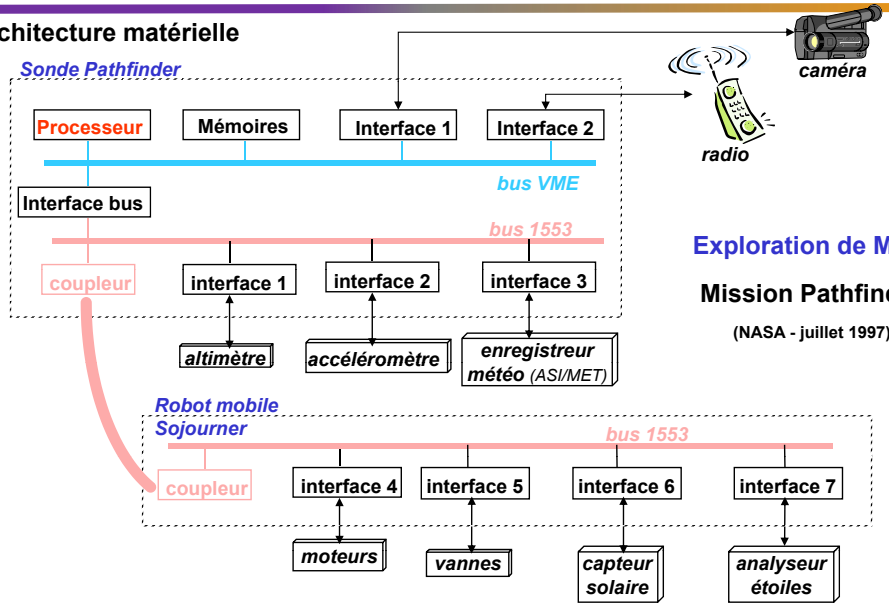
- Exemple de CSPEC :
  - Les process de contrôle sont décrits par un diagramme Etats / Transition
  - L'activation et la désactivation des Process est donnée dans une Table d'Activation



Process / Action	Rendre Monnaie	Distribuer Produit	Obtenir Sélection
Accepter Sélection	O	O	1
Rendre Paiement	1	O	O
Accepter Pièce	O	O	O
Distribuer Produit	1	2	O

Méthode de modélisation
5.2. Exemple : Mission Mars Pathfinder

Architecture matérielle

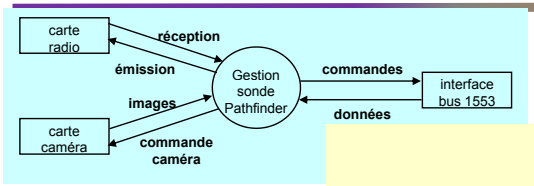


Exploration de Mars

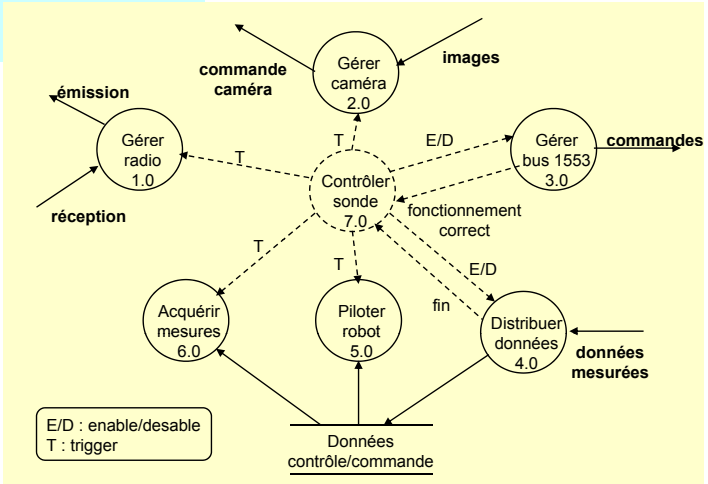
Mission Pathfinder

(NASA - juillet 1997)

Méthode de modélisation
5.2. Exemple : Mission Mars Pathfinder

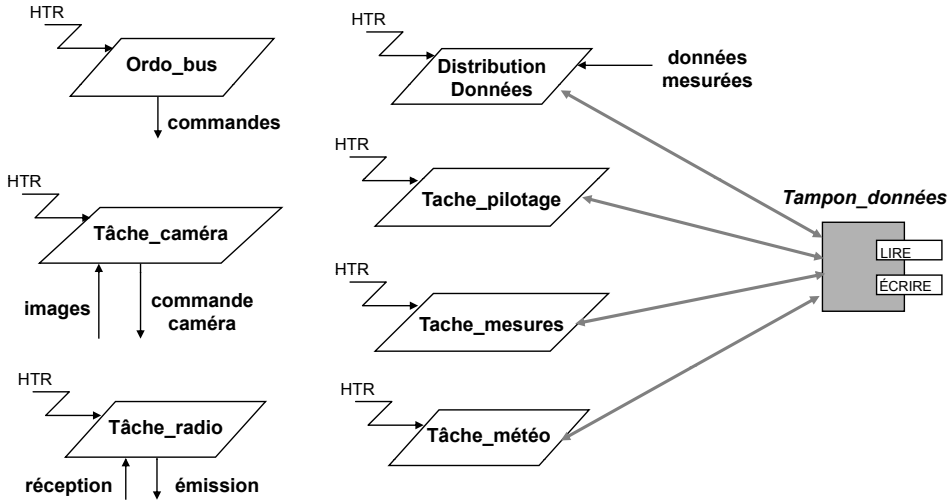


Architecture logicielle



Méthode de modélisation
5.2. Exemple : Mission Mars Pathfinder

Les tâches de l'application en phase d'exploration / 1

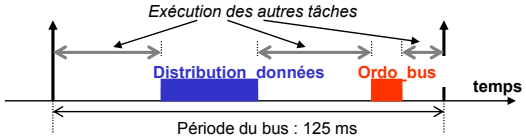


Méthode de modélisation
5.2. Exemple : Mission Mars Pathfinder

Les tâches de l'application en phase d'exploration / 2

Priorité	Tâche	Commentaire
La plus haute	ORDO_BUS	tâche ordonnanceur du bus 1553
↑	DISTRIBUTION_DONNEES	tâche distribution des données du bus 1553
↑	TACHE_PILOTAGE	tâche de pilotage de l'application (robot)
↑	TACHE_RADIO	tâche de gestion des communications radio
↑	TACHE_CAMERA	tâche de gestion de la camera
↑	TACHE_MESURE	tâche dédiée aux mesures
La plus basse	TACHE_METEO	tâche de gestion des données météo

Diagramme temporel de fonctionnement du bus 1553



Tâche	Priorité	Ci	Pi
ORDO_BUS	7	25	125
DISTRIBUTION_DONEES	6	25	125

Méthode de modélisation
5.2. Exemple : Mission Mars Pathfinder

Caractéristiques temporelles des tâches

		Paramètres (ms)		Paramètres réduits	
Tâche	Priorité	Ci	Pi	Ci	Pi
ORDO_BUS	7	25	125	1	5
DISTRIBUTION_DONNEES	6	25	125	1	5
TACHE_PILOTAGE	5	25	250	1	10
TACHE_RADIO	4	25	250	1	10
TACHE_CAMERA	3	25	250	1	10
TACHE_MESURE	2	50	5000	2	200
TACHE_METEO	1	{50,75}	5000	{2,3}	200

Méthode de modélisation
5.2. Exemple : Mission Mars Pathfinder

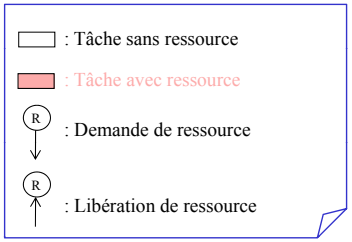
Caractéristiques du noyau temps réel

- ordonnancement préemptif basé sur la priorité fixe des tâches (RM)
- gestion des files d'attente selon la priorité des tâches
- demande et libération des ressources en début et fin d'exécution
- une demande de ressource non satisfaite est de durée nulle

noyau temps réel VxWorks (Wind River Systems)

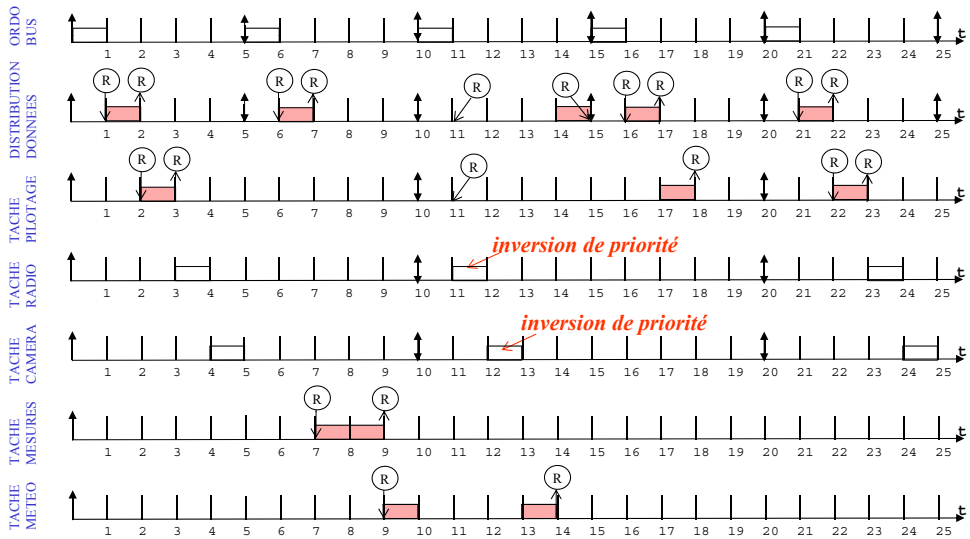
Simulations

Légende des diagrammes de Gantt



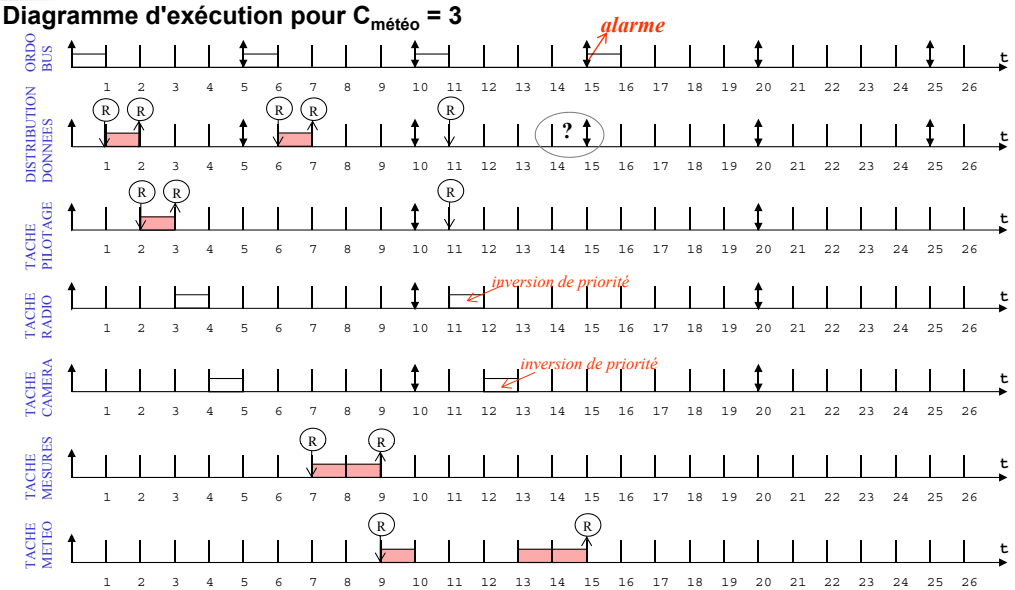
Méthode de modélisation
5.2. Exemple : Mission Mars Pathfinder

Diagramme d'exécution des tâches pour $C_{météo}$ = 2



: Tâche sans ressource : Tâche avec ressource

Diagramme d'exécution pour $C_{météo}$ = 3



Bibliographie

C. BONNET et I. DEMEURE
Introduction aux systèmes temps réel, HERMES, 1999.

J.A. BIANCOLIN
Spécification et conception des systèmes temps réels, HERMES, 1995.

A. DORSEUIL et P. PILLOT
Le temps réel en milieu industriel, DUNOD Ed, 1993.

H. BRENIER
CIM et Temps réel. Mise en œuvre d'un atelier génie logiciel et de génie automatique, DUNOD Ed, 1992.

LERAY Joël
Les bases de l'informatique industrielle, DUNOD Ed, 1992.

J.P. PEREZ
Systèmes temps réels, DUNOD Ed, 1990.

D. TSCHIRHART
Commande en temps réel, DUNOD Ed, 1990.