UNIVERSITÉ TOULOUSE III PAUL SABATIER

Cours de Master recherche (et DEA)

Ordonnancement dans les systèmes temps réel

1999-2018

Zoubir Mammeri

Chapitre 1

Introduction aux systèmes embarqués et systèmes temps réel

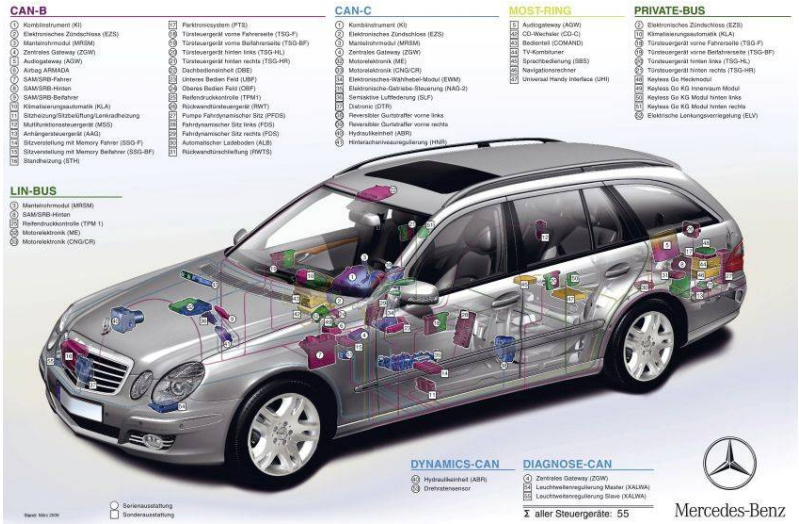
1. Notions et caractéristiques des systèmes embarqués

Domaines d'application des systèmes embarqués

- Automatismes, Armements...
  - Avionique (civil et militaire)
  - Robotique
  - Automobile
  - Villes intelligentes
  - Armement (chars, drones, soldats...)

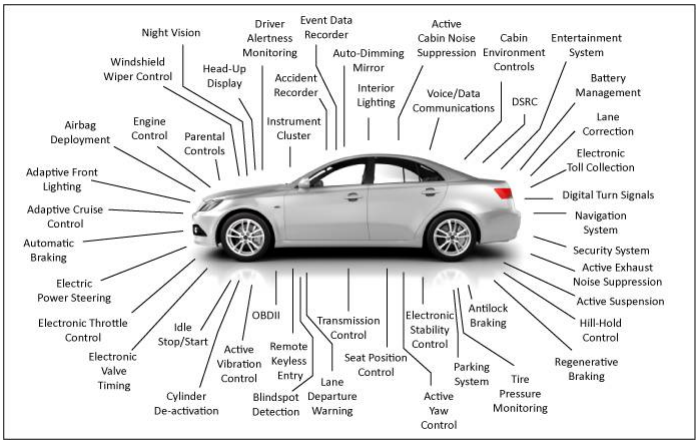
- Multimédia, e-santé, e-commerce...
  - Télécommunications (téléphonie, Internet, e-commerce...)
  - Implants (santé, sécurité...)
  - Informatique diffuse ('pervasive/ubiquitous computing') :
    domotique, immeubles intelligents, 'virtual classroom',...
  - Jeux
  - Vêtements...

Exemple typique de système embarqué

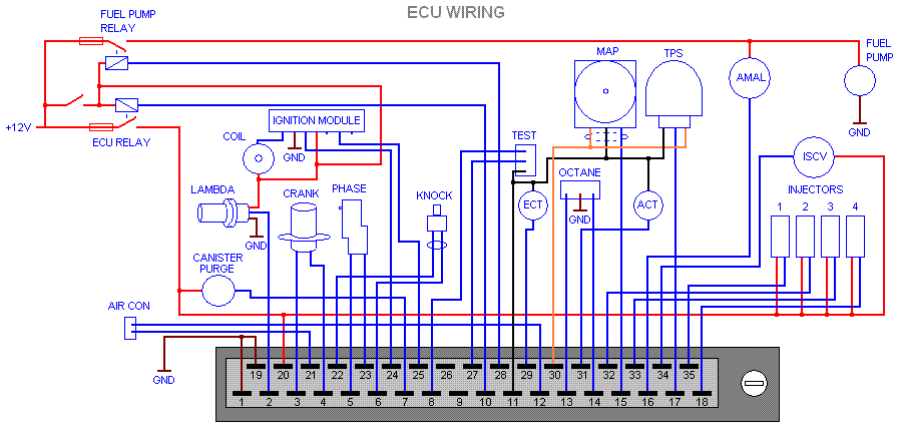


55 ECU (electronic control unit) , 7 Bus de communication

Exemple typique de système embarqué



Exemple typique de système embarqué (Contrôle moteur)



Exemple typique de système embarqué
ECU (Electronic Conrol Units)

Exemples de systèmes embarqués

Définitions

1. Un système embarqué (SE) est un système informatisé spécialisé qui constitue une partie intégrante d'un système plus large ou une machine. Typiquement, c'est un système sur un seul processeur et dont les programmes sont stockés en ROM.

A priori, tous les systèmes qui ont des interfaces digitales (i.e. caméra, voiture…) peuvent être considérés comme des SE. Certains SE ont un système d'exploitation et d'autres non car toute leur logique peut être implantée en un seul programme.

2. Un système embarqué est une combinaison de logiciel et matériel, avec des capacités fixes ou programmables, qui est spécialement conçu pour un type d'application particulier. Les distributeurs automatiques de boissons, les automobiles, les équipements médicaux, les caméras, les avions, les jouets, les téléphones portables et les PDA sont des exemples de systèmes qui abritent des SE. Les SE programmables sont dotés d'interfaces de programmation et leur programmation est une activité spécialisée.

3. Un système embarqué est une composante primordiale d'un système (i.e. un avion, une voiture…) dont l'objectif est de commander, contrôler et superviser ce système.

⇒ **'Embedded' : Enfoui / Embarqué**

Principales caractéristiques

- **Encombrement mémoire (mémoire limitée, pas de disque en général)**
- **Consommation d'énergie (batterie : point faible des SE)**
- **Autonomie**
- **Mobilité**
- **Communication (attention : la communication affecte la batterie)**
- **Contraintes de sécurité**
- **Contraintes de temps réel**
- **Ergonomie**
- **Impacts : éthiques, sociétaux…**
- **Coût de produits en relation avec le secteur cible**

Besoins

- **Intégration massive de composants embarqués répartis pour bâtir la société de l'information :**

  téléphones cellulaires – PDA – machines programmables – automobile - appareils médicaux – photo/vidéo/hi fi – électroménager – avionique - spatial – jouets

- **Outils théoriques et pratiques pour l'intégration permettant de prendre en compte tous ces critères à la fois**
  - **Fonctionnalité**
  - **Qualité : Robustesse et Performances**
  - **Contraintes matérielles (poids, encombrement, ..)**
  - **Coût global, consommation électrique**
  - **Délai de mise sur le marché**

- **Construire des systèmes de fonctionnalité et qualité déterminée et garantie, à coût acceptable, est un défi technologique et scientifique majeur.**

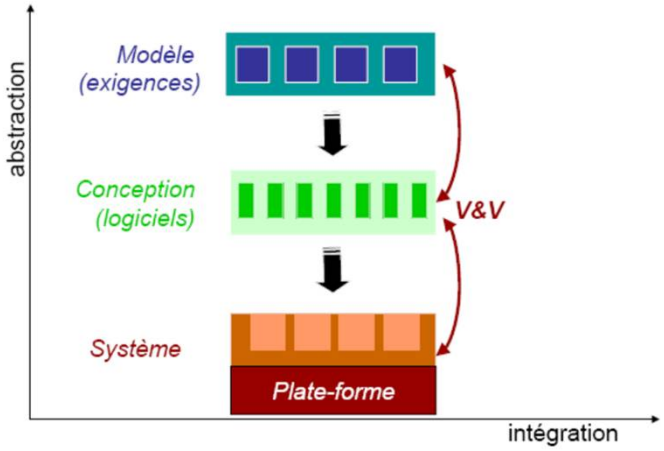
Aspects à considérer

- **Techniques : Conception conjointe (Matériel, Logiciel, Environnement)**
- **Économiques : Optimisation par rapport au marché, entre coût et qualité**
- **Multi-compétence : Combinaison de compétences en logiciel, contrôle, réseaux, ingénierie en électronique, IHM, automatique, médecine, mécanique...**

Défis à considérer

- **Hétérogénéité : Construire des systèmes complexes par intégration de composants hétérogènes (mécanismes de communication, vitesse de fonctionnement, granularité des calculs, variété des médias...)**
- **Complexité : L'effort de développement augmente exponentiellement avec le nombre de composants intégrés. D'où la nécessité de remplacer les méthodes de validation a posteriori par des méthodes de validation incrémentale. Développer des outils théoriques et techniques pour la validation incrémentale.**
- **Intelligence : Moyen d'améliorer la qualité (robustesse et la performance) de système : Autodiagnostic, Auto configuration, Adaptabilité à l'environnement, Evaluation des risques...**

Développement de système embarqué



2. Notions et caractéristiques des systèmes temps réel

La gestion du temps dans un OS classique (temps partagé)

- **Un système d'exploitation (OS) classique, c.-à-d. orienté temps partagé, doit organiser et optimiser l'utilisation des ressources de façon à ce que l'accès à ces ressources soit équitable. Un OS n'a donc pour seule contrainte de temps que celle d'un temps de réponse satisfaisant pour les utilisateurs avec lesquels il dialogue. Les traitements qu'on lui soumet sont effectués en parallèle au fil de l'allocation des quanta de temps aux diverses applications.**

- **Un OS est capable de dater les événements qui surviennent au cours du déroulement de l'application! : mise à jour d'un fichier, envoi d'un message, etc. Il permet aussi de suspendre un traitement pendant une certain délai, d'en lancer un à une certaine date...**

- **En aucun cas, un OS classique ne garantit que les résultats seront obtenus pour une date précise, surtout si ces résultats sont le fruit d'un traitement déclenché par un événement EXTERIEUR (émission d'un signal par un périphérique quelconque,...)**

Définitions

Notion du temps réel

Temps réel par opposition au **temps logique**

Temps réel → temps physique / temps mesurable

Systèmes temps réel → avec des contraintes exprimées à l'aide du temps physique

Définitions

1. « Intervalle de temps compatible avec le rythme réel d'arrivée des données et à l'intérieur duquel un ordinateur peut effectuer les traitements nécessaires » Le petit Robert.
2. Un résultat juste mais hors délai est un résultat faux.
3. … répondre en temps réel à des stimuli ...
4. « A real-time system is defined as a system whose correctness of the system depends not only on the logical results of computations, but also on the time at which the results are produced » [Stankovic 1988].

Exemples de contraintes temps réel

- **Prélever la température toutes les 50 secondes.**
- **Déclencher l'airbag au plus tard 10 ms après l'impact.**
- **Envoyer 25 images par seconde**
- **Lancer la sirène de l'usine à midi**
- **Mettre à jour l'état du stock au plus tard 10 min après la vente de produit.**

Exemples de comportements « non temps réel »

- **Prendre 48h pour prédire le temps qu'il fera le lendemain**
- **Calculer un seuil de vente ou d'achat d'actions au bout de 7 jours**
- **Arriver à 9h pour prendre le train de 8h32**

Confusions/ambiguïtés

- Temps réel non strict = absence de contraintes FAUX
- Optimisation des temps de réponse ⇒ Temps réel FAUX
- Temps réel = rapidité FAUX

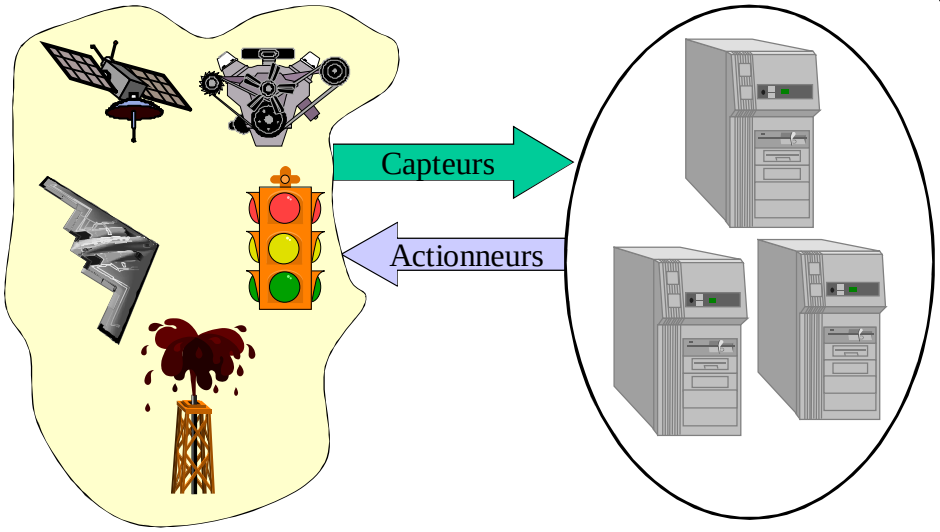
Durée d'exécution

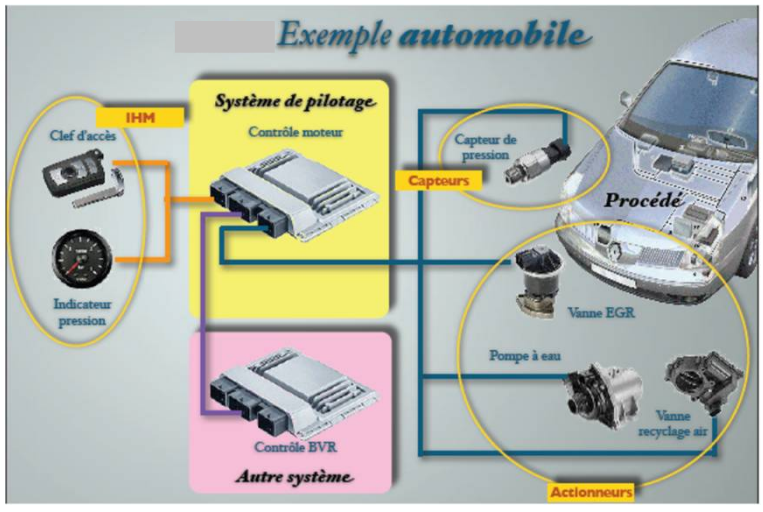
Opération O1	30	10	8	11	14
Opération O2	7	10	11	5	4
Opération O3	8	10	11	11	3
Délai moyen	15	10	10	9	7

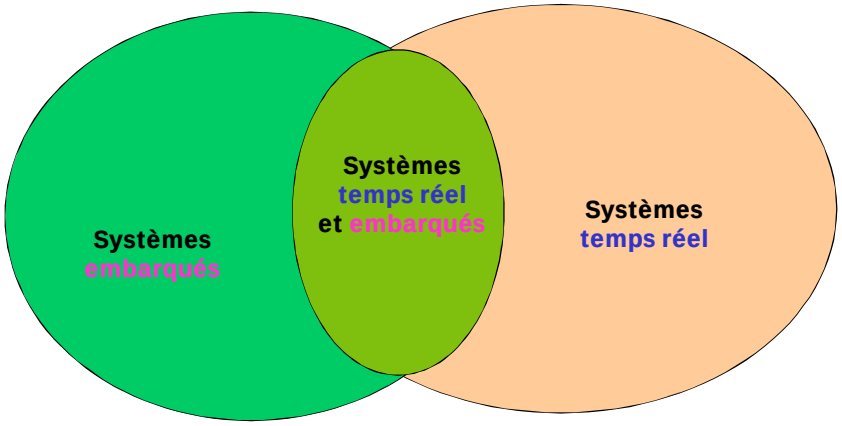
Les opérations O1, O2 et O3 ont une échéance relative égale à 10

Domaines d'applications

- Installations industrielles (chimique, nucléaire, automobile, …)
- Contrôle et régulation de trafic en milieu urbain
- Systèmes embarqués (voitures, trains, ...)
- Télécommunications
- Avionique et aérospatial
- Domaine militaire
- Multimédia et Web (téléconférences, télé-achat, …)
- Domotique, Jeux
- Médecine, Télé-médecine
- Réalité virtuelle, Travail coopératif
- Autres


Capteurs
Actionneurs
Environnement
Système de contrôle


Exemple automobile
IHM
Clef d'accès
Indicateur pression
Système de pilotage
Contrôle moteur
Capteurs
Capteur de pression
Procédé
Vanne EGR
Pompe à eau
Vanne recyclage air
Actionneurs
Contrôle BVR
Autre système



Systèmes embarqués $\Rightarrow$ Systèmes temps réel

$\Leftarrow$

Problèmes à résoudre

- Structure dynamique (anomalies, modes de marche)
- Réactivité aux événements non nécessairement prédictibles
- Concurrence
- Communication
- Distribution
- Sécurité

En présence de Contraintes de temps

TR → Tous les problèmes de l'informatique classique
+ les contraintes temporelles

L'ordonnancement joue un rôle fondamental

Systèmes mono-utilisateur monotâche

Systèmes mono-utilisateur multitâches

Système multi-utilisateurs multitâches

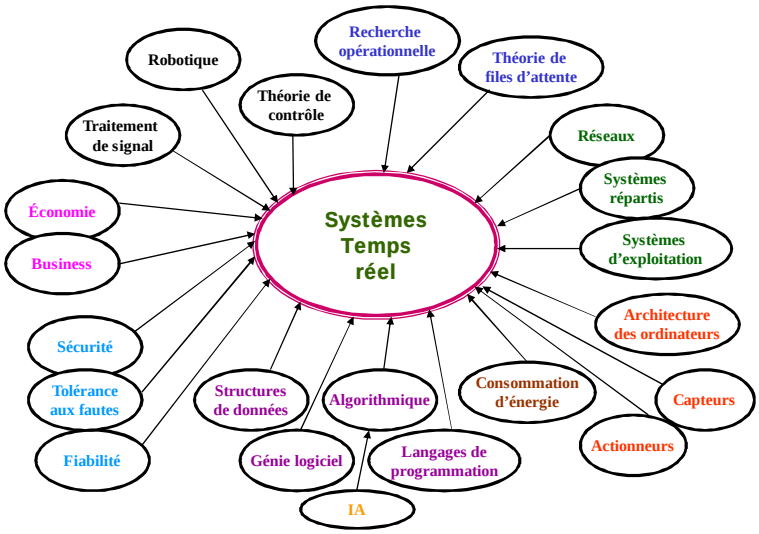
Systèmes distribués

Systèmes temps réel centralisés

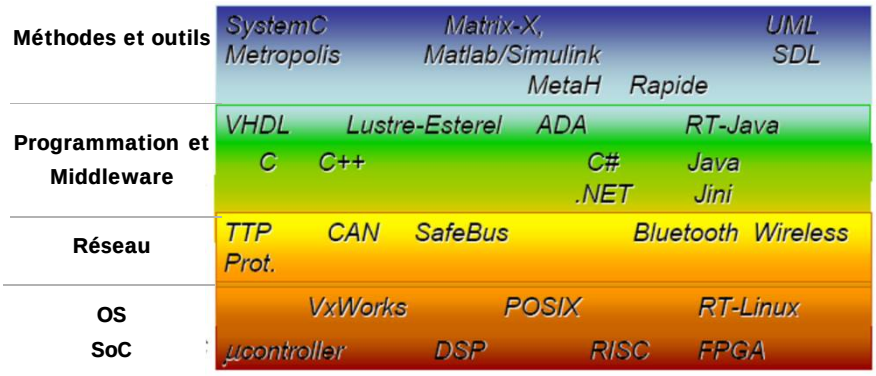
Systèmes temps réel et distribués

Complexité

Disciplines impliquées dans les STR



Intégration de technologies



3. Contraintes de temps

Différentes perceptions du temps réel (1/3)

➔ **Temps vu par celui qui spécifie**

Objectif : Fixer les contraintes temporelles pour que le SI respecte les caractéristiques/propriétés de l'environnement

Déterminer :

- **Echelle du temps : sec, ms, …**
- **Intervalle minimal pour distinguer les événements**
- **Durée de validité des événements et données**
- **Temps de réponse aux stimuli**

Différentes perceptions du temps réel (2/3)

➔ **Temps vu par celui qui vérifie**

Objectif : Vérifier des propriétés

- Sûreté (Safety)
- Vivacité (Liveness)
- Equité (Fairness)

Temps réduit à une succession d'événements

Communauté de la logique temporelle

- temps linéaire
- temps arborescent

Pour étudier les propriétés d'un programme

Différentes perceptions du temps réel (3/3)

➔ **Temps vu par celui qui implante**

Objectif : garantir le respect des CT par l'architecture support

Choisir :

Temps physique (horloge)

Temps absolu ou relatif

Temps local et temps global

Algorithmes d'ordonnancement

Sources des contraintes de temps (1/2)

➔ **Origines externes**

- **Caractéristiques physiques (vitesse, durée de réaction chimique, …)**
- **Caractéristiques liées aux lois de commande du système physique**
- **Qualité de service requise en terme de délai de réponse tolérable (qualité audio/vidéo)**
- **Perception sensorielle et le temps de réaction de l'homme**
  - **. Espacement des signaux envoyés à un pilote pour atterrir ou changer de trajectoire (durée minimale des signaux pour que le pilote puisse les voir)**
  - **. Un signal d'alarme doit durer au moins un temps min (pour être vu/entendu)**
- **Contraintes à caractère commercial**
- **Autres**

Connaissance des aspects physiques nécessaire

Sources des contraintes de temps (2/2)

➔ **Origines internes**

- **Choix de conception**
  - **architecture centralisée ou répartie (données, traitements, contrôle)**
  - **actions périodiques (avec les périodes adéquates) ou apériodiques**
  - **choix d'une structure d'application (interface entre composants, …)**
  - **autres**
- **Choix d'architecture matérielle et logicielle**
  - **processeurs avec des vitesses particulières**
  - **système d'exploitation (intégrant des algorithmes d'ordonnancement)**
  - **réseau avec des débits et des temps de réponse donnés**
  - **autres**

Expression des contraintes de temps (1/4)

Expression à l'aide du temps quantifié (temps physique)

➔ **Temps absolu**

– **un instant (date)**

Ex. La sirène doit sonner trois fois à midi le premier mercredi de chaque mois.

– **une fenêtre temporelle (intervalle de temps)**

Ex. Produire les pièces du modèle Z de 8 h à 12 h.

➔ **Temps relatif**

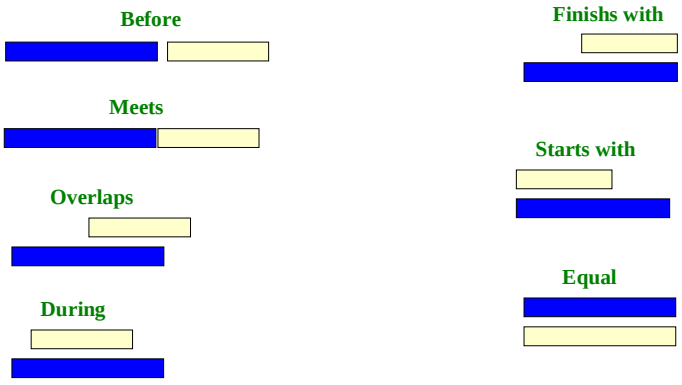
– **une durée (délai, période), minimale, maximale, moyenne**

Ex. Délai de transfert de message min = 10 ms, max = 100 ms

Expression des contraintes de temps (2/4)

Expression à l'aide du temps non quantifié

➔ **Relations sur les intervalles de temps : Opérateurs d'Allen**



Expression des contraintes de temps (3/4)

Expression sur les événements

- **Contraintes sur la durée de vie d'un événement**
  Ex. La porte de l'ascenseur doit rester ouverte pendant 15 secondes après ouverture.
- **Contraintes sur les instants et l'ordre d'occurrence d'événements**
  Ex. L'ordre d'ouverture de l'airbag doit être donné au plus tard 2 ms après l'impact.

Expression sur les données

- **Durée de validité (ou contrainte de fraîcheur)**
  Ex. Les mesures de température sont valides pendant 100 ms au plus.
- **Contraintes de production**
  Ex. Donner la position du mobile toutes les 10 secondes.
- **Contraintes de corrélation**
  Ex. Température et pression doivent être mesurées au maximum à 10 ms d'intervalle.

Expression des contraintes de temps (4/4)

Expression sur les actions (opérations)

- **Période** (ex. toutes les 40 ms rafraîchir l'image)
- **Instant au plus tôt/tard, pour démarrer une action**
- **Instant au plus tard/tôt, pour finir une action**
- **Temps maximum d'exécution (ou échéance relative)**
- **Temps maximal d'attente d'une ressource**
- **Temps minimal/maximal d'utilisation d'une ressource**
- **Temps minimum d'exécution affecté à une action**
- **Précédence entre actions**

Quelques ordres de grandeur

Domaines d'applications	Ordre de grandeur des CT
➔ Processus chimique	en heure / minute
➔ Fabrication	en minute / seconde
➔ Robotique	en 10 ms
➔ Systèmes vocaux	en ms
➔ Systèmes radar	en ms
➔ Contrôle moteur	en µs

Temps réel $\not\Rightarrow$ Rapidité

$\not\Leftarrow$

Criticité et fermeté des CT
(criticalness and firmness)

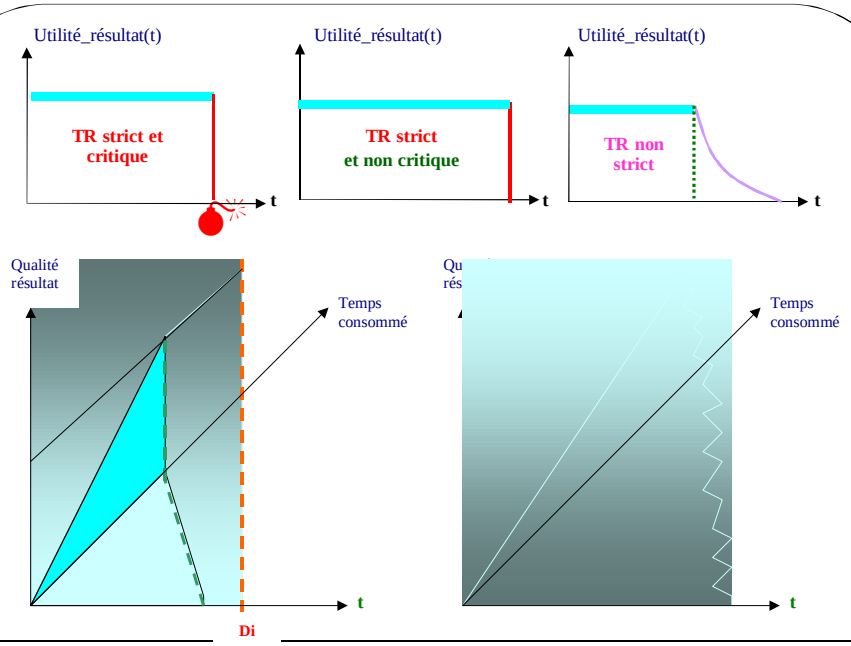
➔ **Temps réel critique ou non (Hard RTS or Soft RTS)**

⇒ **Conséquences des fautes temporelles**

- sur la sûreté humaine
- sur l'environnement
- mission du système (perte de production….)

➔ **CT strictes ou non strictes (firm and non-firm constraints)**

⇒ **Validité du résultat dans le temps**



Prévisibilité ("predictability")

Prévisibilité : pouvoir démontrer que le système respectera les contraintes aussi longtemps que les hypothèses initiales sont respectées

➔ **Méthodes formelles ont un rôle de première importance à jouer**

Prévisibilité

- **Absolue (garantie à 100%)** ➔ **connaissances statiques des contraintes**

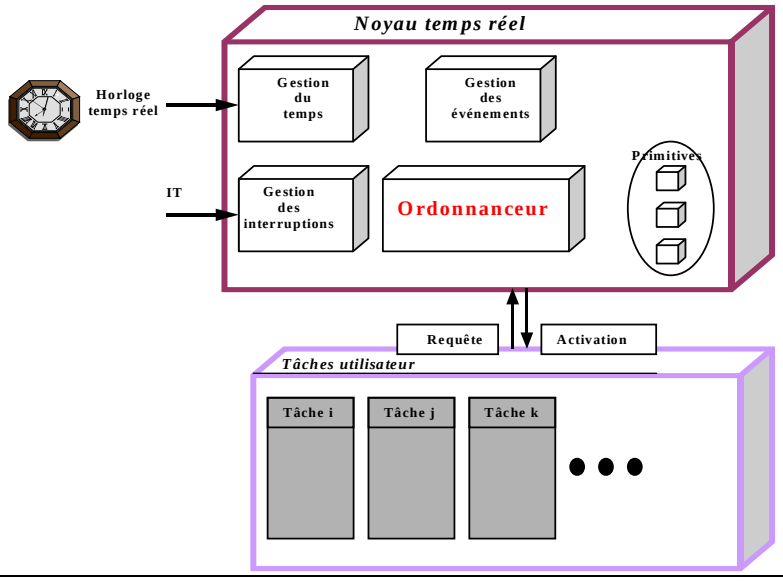
- **Probabiliste**
- **Stochastique**

} ➔ **Accepter que certaines opérations ratent leurs échéances parfois**

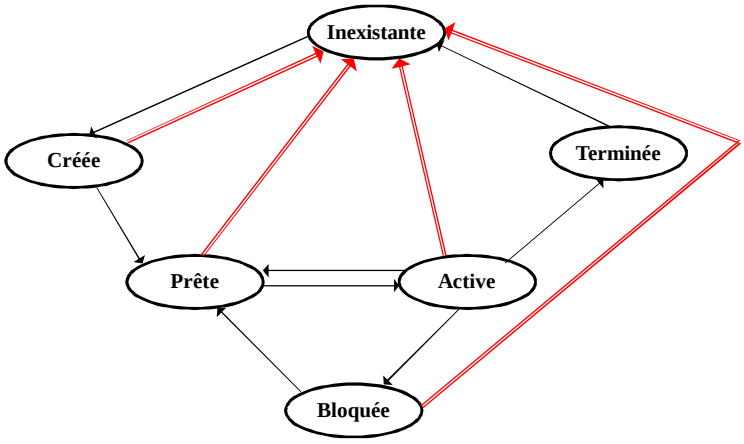
"Any realistic model of a real-world phenomenon must take into account the possibility of randomness. That is, more often than not, the quantities we are interested in will not be predictable in advance but, rather, will exhibit an inherent variation that should be taken into account by the model." [Ross 1972]

Meilleur effort (best effort) : on fait ce que l'on peut

4. Architecture logicielle

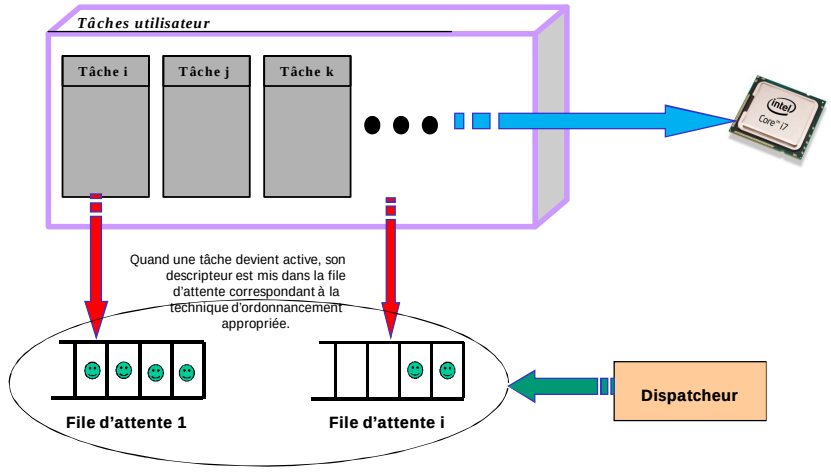


Etats d'une tâche



Ordonnancement dans l'OS

➔ **Composants de l'ordonnanceur : File(s) d'attente + Dispatcheur**



Ordonnancement dans l'OS

➔ **Le dispatcheur prend la main (s'exécute)**

- À l'initialisation (déterminer la première tâche à exécuter)
- Quand la tâche en cours se termine (i.e. pour changer de tâche)
- Quand la tâche en cours se bloque à cause d'une ressource occupée
- Quand le quantum de temps de la tâche en cours est fini (cas du temps partagé)
- Quand une nouvelle tâche devient prête (ce passage d'état se fait par le gestionnaire de ressource, ou par le *timer* lié aux tâches périodiques)

➔ **Le dispatcheur utilise les niveaux de priorité natifs (s'il y en a)**

- Le nombre de niveaux de priorité natifs est limité selon les OS
- Attention à l'ordre des niveaux
  - 0 < 1 < 2 ... < 255 (ordre numérique)
  - 0 > 1 > 2 ... > 255 (ordre numérique inversé)

➔ **L'OS peut inclure ou non des primitives de changement de priorité de tâche en cours d'exécution**

Ordonnancement dans l'OS

➔ **Techniques d'ordonnancement de base**

– Sans priorité

* FIFO (ordonnancement non préemptif par nature)

+ Le dispatcheur alloue le processeur à a tâche en tête de file d'attente jusqu'à ce qu'elle se termine avant de passer à la tâche suivante.

* Round Robin (ordonnancement préemptif de nature)

+ Le dispatcheur alloue le processeur à la tâche en tête de file d'attente pendant un laps de temps (le laps de temps est identique pour toutes les tâches ou variable selon les tâches).

+ Si la tâche se termine pendant son laps de temps, le dispatcheur est activé pour sélectionner la tâche suivante.
Sinon le timer de décompte le temps alloué à la tâche se déclenche et la tâche est recyclée en queue de file. Ensuite, le dispatcheur sélectionne la tâche suivante.

Ordonnancement dans l'OS

➔ **Techniques d'ordonnancement de base (suite)**

– Avec priorité (ordonnancement préemptif ou non préemptif)

* Statique (la tâche s'exécute toujours avec le même niveau de priorité)

+ sans préemption :
Le dispatcheur sélectionne la tâche la plus prioritaire parmi les tâches prêtes et lui alloue le processeur. Elle s'exécute jusqu'à la fin. Ensuite, le dispatcheur est activé pour sélectionner la tâche prioritaire suivante.

+ avec préemption :
Le dispatcheur sélectionne la tâche la plus prioritaire parmi les tâches prêtes et lui alloue le processeur. Le dispatcheur est activé quand une nouvelle tâche demande à s'exécuter et peut interrompre la tâche en cours si une des tâches qui viennent d'arriver est plus prioritaires. La tâche interrompue est remise en file d'attente.

* Dynamique (la tâche peut changer de niveau de priorité en cours d'exécution)

OS temps réel

➔ **Mécanismes/Fonctions exigés d'un noyau temps réel**

- Gestion de tâches (création, destruction...)
- Ordonnancement
- Synchronisation de tâches (mutex, sémaphores...)
- Communication (locale et distante)
- Gestion du temps (horloge temps réel)
- Gestion des interruptions
- Gestion (protection) mémoire

➔ **Classes d'OS pour le temps réel**

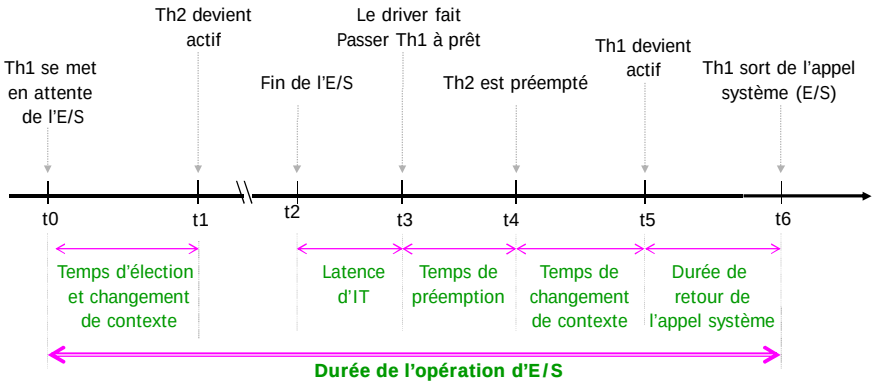
- OS Conventionnels (Unix, Lunix, Windows) : pas de garantie
- Extensions d'OS conventionnels (RT-Linux)
- RTOS (OSEK-VDX, Lynx OS, QNX...)
- OS dédiés (écrits pour les besoins de systèmes spécifiques) :
  une tâche qui active les autres, table de scrutation...

Eléments constituant le temps de réponse d'une tâche

- **Temps d'exécution (Worst Case Execution Time)**
- **Temps d'attente de ressources**
- **Temps d'attente des E/S**
- **Temps dus au système d'exploitation et difficilement calculables**
  - Temps de latence des interruptions
    (délai entre l'arrivée d'une IT et le début de sa prise en compte effective)
  - Temps de préemption
    (délai de sélection de la tâche à exécuter)
  - Latence de l'ordonnancement
    (délai entre la réception de l'IT et l'entrée dans la tâche de l'utilisateur)
  - Temps de commutation de contexte de tâche
  - Latence de sémaphore
    (délai entre la libération d'un sémaphore et la réactivation de la tâche bloquée derrière ce sémaphore)

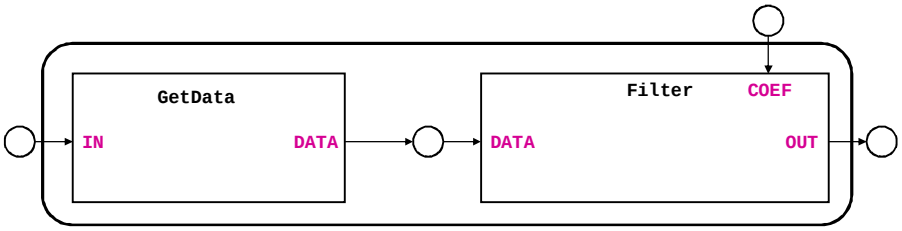
Exemple de temps de réponse à un événement

- **On considère deux threads Th1 et Th2. Th1 est le plus prioritaire.**
- **A l'instant t0, Th1 lance une E/S.**
- **Le dessin ci-dessous schématise la suite des événements :**



Exemple de tâches temps réel (1/2)

(en langage FlowC)



La tâche `GetData` mesure une température à partir de l'environnement et envoie la valeur mesurée à la tâche `Filter`.

Une mesure est effectuée toutes les 10 ms.

Quand `N` mesures ont été effectuées, la moyenne des `N` mesures est envoyée à `Filter`.

La tâche `Filter` lit `N` mesures et les ignore et lit ensuite la moyenne des mesures, multiplie cette moyenne par `c` (dont la valeur est lue à partir de `COEF`) et envoie la valeur obtenue à l'environnement via le port `OUT`.

Exemple de tâches temps réel (2/2)

(en langage FlowC)

```
Process GetData
(InPort IN, OutPort DATA)
{ float mesure, somme; int i;
  While(1)
  { somme = 0;
    for (i=0; i< N; i++) {
         READ(IN, mesure ,1);
         somme+= mesure ;
         WRITE(DATA, mesure, 1)
         DELAY(10)
                          }
   WRITE(DATA, somme/N,1)
}}
```

```
Process Filter
(InPort DATA, InPort COEF, OutPort OUT)
{ float c, d; int j;
  c = 1; j = 0;
  While(1)
  { SELECT(DATA, COEF) {
    CASE DATA: READ(DATA, d, 1);
         if (j==N) {j=0; d=d*c;
                   WRITE(OUT, d,1); }
         else j++;
   CASE COEF: READ(COEF, c, 1); break ;
}}}
```

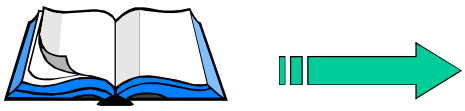
Temps réel ou pas temps réel ? A discuter

- **Acquisition de données en temps réel**
- **Gestion de stock en temps réel**
- **Traitement d'images en temps réel**
- **Trafic des bus en temps réel**
- **Base de données temps réel**
- **Système avec réponse immédiate**
- **Système très rapide**
- **Système réactif**
- **Système critique**
- **Système avec temps de réponse borné**
- **Etc.**

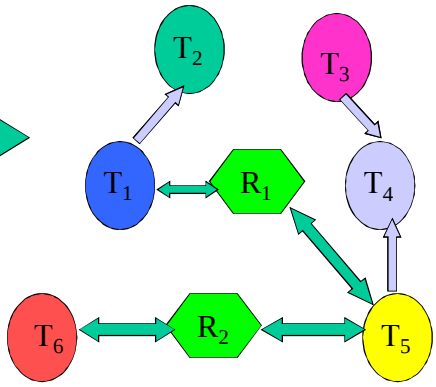
Chapitre 2

Généralités sur l'ordonnancement de tâches

1. Modèles de tâches (contraintes)

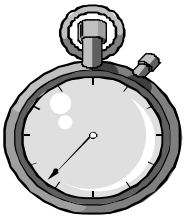


Document de conception

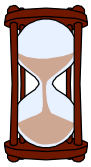


Ensemble de tâches

Contraintes de tâches (1/5)



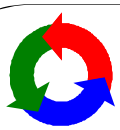
Top départ



Durée d'exécution

Echéance

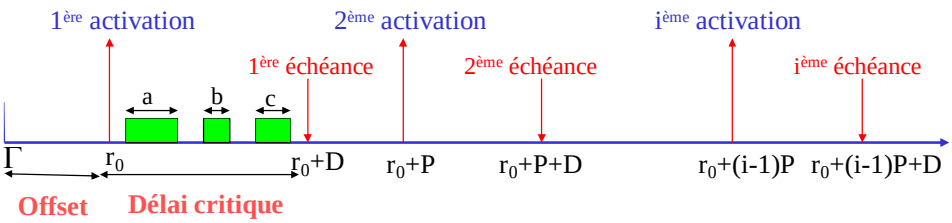
- **individuelles**
- **collectives**



Contraintes de tâches (2/5)

Modèle simple (tâches périodiques)

Mesure de température



Offset **Délai critique**

Γ: instant initial (démarrage de l'application)

r_0 : date de première activation

C : pire durée d'exécution = Max(a) + Max(b) + Max(c)

D : délai critique

P : période **Si P = D : échéance sur requête**

Charge-processeur d'une application TR

$u_i = \frac{C_i}{P_i}$: **Pourcentage de l'activité du processeur dédiée à la tâche T_i**

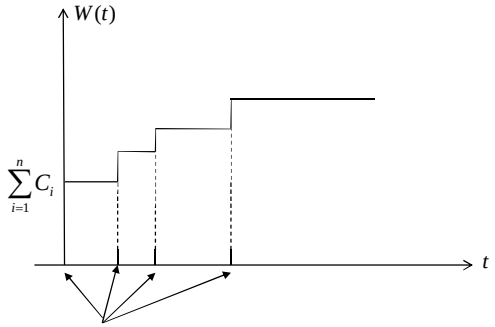
$U = \sum_{i=1}^{n} \frac{C_i}{P_i}$: **Charge processeur = taux d'utilisation du processeur pour les tâches périodiques**

U > m ⇒ application non ordonnançable sur m processeurs
(U ≤ m : Condition nécessaire, mais pas suffisante)

Demande de processeur

Quantité de travail totale demandée au processeur à l'instant *t : W(t)*

$$W(t) = \sum_{i=1}^{n} \left(1 + \left\lfloor \frac{t}{P_i} \right\rfloor\right) \times C_i$$



Instants de demande d'activation de tâche

Plus longue période d'activité du processeur
Cas de tâches périodiques

Pire cas = cas où toutes les tâches sont activables à l'instant t=0

Plus longue période d'activité du processeur L est donnée par :

$$L = \sum_{i=1}^{n} \left\lceil \frac{L}{P_i} \right\rceil \times C_i$$

Point fixe ⇒ calcul itératif

$$\begin{cases} L^{m+1} = W\left(L^m\right) \\ L^0 = \sum_{i=1}^{n} C_i \end{cases}$$

On s'arrête quand $L^{m+1} = L^m$

Plus longue période d'activité du processeur
Cas de tâches périodiques

Exemple de calcul de L

	C_i	P_i
T_1	7	20
T_2	5	20
T_3	8	30
T_4	3	100
T_5	2	100

$$L^0 = \sum_{i=1}^{n} C_i = 7+5+8+3+2 = 25$$

$$L^1 = W(L^0) = \sum_{i=1}^{n} \left(1 + \left\lfloor \frac{25}{P_i} \right\rfloor\right) \times C_i =$$

$$(2*7)+(2*5)+(1*8)+(1*3)+(1*2) = 37$$

$$L^2 = W(L^1) = \sum_{i=1}^{n} \left(1 + \left\lfloor \frac{37}{P_i} \right\rfloor\right) \times C_i = 45$$

$$L^3 = W(L^2) = \sum_{i=1}^{n} \left(1 + \left\lfloor \frac{45}{P_i} \right\rfloor\right) \times C_i = 57$$

$$L^4 = W(L^3) == \sum_{i=1}^{n} \left(1 + \left\lfloor \frac{57}{P_i} \right\rfloor\right) \times C_i = 57$$

Contraintes de tâches (3/5)

Modèle “complet” de tâche (contraintes individuelles)

- Type de tâche (critique ou non)
- Importance de tâche et Priorité de tâche
- Autorisation de réquisition du processeur
- Contraintes temporelles
  - Tâche périodique, sporadique ou apériodique
  - Période (P_i)
  - Instant d’arrivée (r_i)

    Fixe = r_i

    Dans un intervalle [r_i^{min} , r_i^{max}]

    Cyclique = K*P + r_0

    Avec Offset (fixe ou dynamique)

    Aléatoire

    sporadique (Intervalle d’interarrivée minimal connu)

Contraintes de tâches (4/5)

Modèle “complet” de tâche (contraintes individuelles)

- Durée d'exécution (C_i)

  Maximale (Worst Case Execution Time)

  Dans un intervalle [C_i^{min} , C_i^{max}]

  Moyenne, probabiliste, etc.

- Echéance absolue (date au plus tard de terminaison) (d_i)
- Echéance relative (temps de réponse maximum) (D_i)
- Terminaison (totale, partielle, …)
- Gigue d’activation, de terminaison…
- Autres

- Contraintes liées à la précision des résultats des tâches

Contraintes de tâches (5/5)

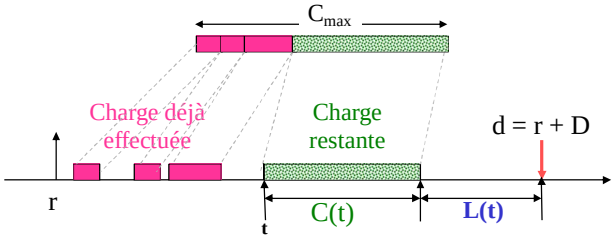
Modèle "complet" de tâche (contraintes collectives)

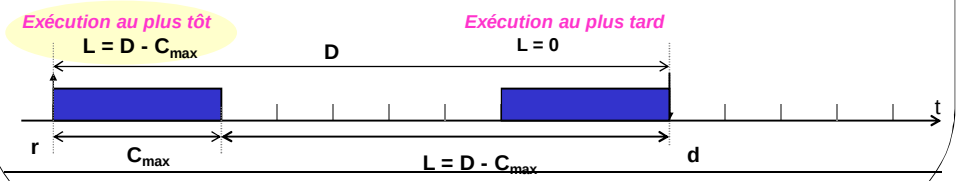
- **Contraintes de relations entre tâches**
  - **Précédence**
  - **Partage de ressources**
- **Contraintes de répartition**
- **Contraintes de tolérance aux fautes**

Paramètres dynamiques de tâche (1/4)

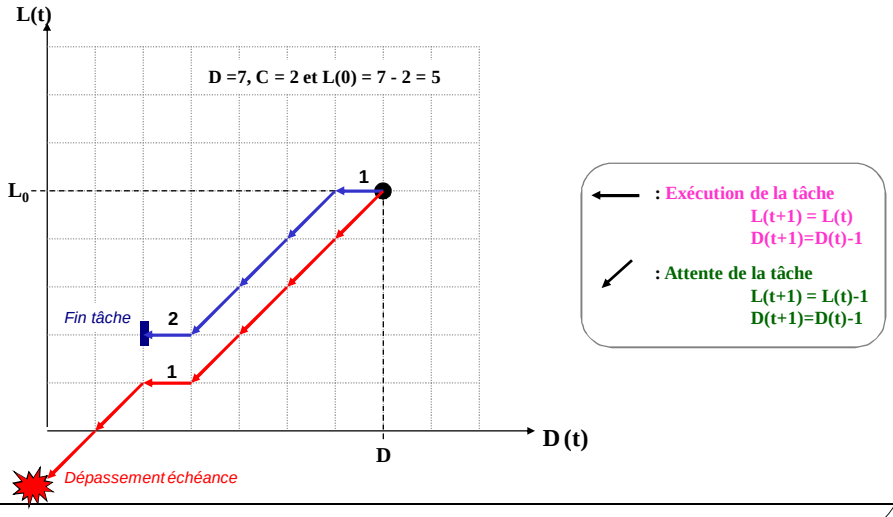
- **temps d'exécution restant C(t)** : $C(t) = C_{max} - C_{consommé}(t)$
- **laxité dynamique L(t)** : temps avant le début d'exécution sans interruption de la tâche

$L(t) = d - t - C(t) = r + D - t - C(t)$



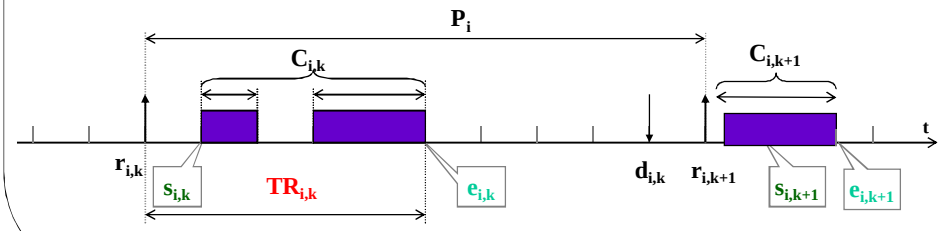


Paramètres dynamiques de tâche (2/4)



Paramètres dynamiques de tâche (3/4)

- **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 : $TR_{i,moy} = \Sigma_k\{TR_{i,k}\}/k$



Paramètres dynamiques de tâche (4/4)

➔ **Gigue absolue max d'activation de T_i :**

$$GA_i = max_k(s_{i,k} - r_{i,k}) - min_k(s_{i,k} - r_{i,k})$$

➔ **Gigue absolue max de terminaison de T_i :**

$$GT_i = max_k(e_{i,k} - r_{i,k}) - min_k(e_{i,k} - r_{i,k})$$

Validation d'applications TR critiques

- Identifier à partir des spécifications la classe de problèmes à résoudre
- Déterminer les scénarii pire cas pour ce problème et exprimer les conditions de faisabilité (CF) associées
- Donner les valeurs numériques des paramètres des CF
- Vérifier que les CF sont validées

Problème : Il y a très peu de conditions de faisabilité pour répondre aux besoins réels.
⇒ Recherche active pour trouver des CF.

2. Classes d'algorithmes d'ordonnancement

- **Ordonnancement = liste (partiellement) ordonnée d'accès aux ressources partagées**
- **Entités ordonnançables : tâches, threads, processus, messages, …**
- **Ordonnancement valide : Ordonnancement qui satisfait les CT**

Algorithmes d'ordonnancement classiques

- **FIFO (à priorités dynamiques)**
  - Les tâches sont traitées dans l'ordre de leur demande d'activation.
  - Toutes les tâches ont la même importance pour FIFO.
  - FIFO minimise le temps de réponse de l'ensemble des tâches.
  - FIFO n'est pas optimal.
- **Round Robin (à priorités dynamiques)**
  - Le processeur est attribué à tour de rôle aux tâches actives.
  - RR est équitable.
  - RR n'est pas optimal pour des tâches temps réel.

FIFO et RR ne garantissent pas les contraintes temporelles dans le cas général

Critères de classification des algorithmes (1/3)

Ordonnancement statique

- Connaissance statique de tous les paramètres de tâches
- Adapté à des systèmes « figés »

Beaucoup de résultats pour l'analyse de l'ordonnançabilité

Prédictibilité : Oui

Ordonnancement dynamique

- Connaissance des paramètres de tâches au moment de leur arrivée
- Tient compte des changements ou événements

Peu de résultats pour l'analyse de l'ordonnançabilité

Prédictibilité : NON

Ordonnancement hors ligne

- Calcul préalable d'une séquence ordonnancement suivie en ligne par un dispatcheur

Ordonnancement en ligne

- Elaboration de séquence d'ordonnancement à l'exécution

Critères de classification des algorithmes (2/3)

Préemptif	Non préemptif
Non ordonnançable en préemptif ⇏	Non ordonnançable en non préemptif
Dirigé par une table statique	Dirigé par les priorités
A priorités statiques (FP)	A priorités dynamiques (DP)
Optimal (dans sa classe FP ou DP) — S'il trouve toujours une solution faisable lorsqu'il en existe une.	Non optimal
Stable — Si on diminue la demande processeur, l'application reste ordonnançable.	Instable

Critères de classification des algorithmes (3/3)

Monoprocesseur	**Multiprocesseur**
Local	**Global** (distribué)
Oisif Retarde des décisions même si la file n'est pas vide (utile en milieu décentralisé)	**Non oisif** Pas de **temps creux** si une tâche est prête

Autres critères de classification des algorithmes (1/2)

- **Complexité**
- **Tolérance aux fautes**
- **Critère optimisé**
  - **Nombre de tâches ne respectant pas les CT**
  - **Longueur de l'ordonnancement**
  - **Equilibrage de charge entre les processeurs**
  - **Charge de communication**
  - **Erreur accumulée** (tâches incrémentales/résultat imprécis)
  - **Autre**

Autres critères de classification des algorithmes (2/2)

- **Technique utilisée**
  - **Théorie des graphes**
  - **Recuit simulé**
  - **Algorithmes génétiques**
  - **Réseaux de neurones**
  - **Théorie des files d'attente**
  - **Logique floue**
  - **Autre**

Rappels sur la complexité des algorithmes

- **Algorithme en O(f(n))** :
  - O(log(n)) : logarithmique
  - O(n) : linéaire
  - O(n^k) : polynomial
  - O(k^n) : exponentiel
- **Problème polynomial (P)** : problème pouvant être résolu en un temps polynomial
- **Classe P** : ensemble des problèmes P
- **Problème non polynomial (NP) :** problème ne pouvant pas être résolu en un temps polynomial
- **Classe NP** : ensemble des problèmes NP
- **Problème NP-complet ('NP-complete')** : un problème *p* appartenant à NP est NP-complet si tout autre problème de NP peut être transformé en *p* en un temps polynomial.
- **Problème NP-difficile ('NP-Hard')** : un problème *p* est NP-difficile si tout problème appartenant à NP se transforme en *p* en un temps polynomial, mais on ne sait pas démontrer si *p* appartient à NP.

Quelques propriétés des problèmes

- **Problème faisable : Existence d'un ordonnancement valide**
- **Condition de faisabilité : CF(P) ⟹ P ordonnançable**
- **Complexité du problème : polynomial (P), NP-complet, NP-difficile**
- **Complexité des solutions :**
  - ➔ **Pour certains problèmes polynomiaux : on connaît des solutions optimales de complexité polynomiale (utilisables en ligne)**
  - ➔ **Pour certains problèmes non polynomiaux : on connaît des solutions optimales de complexité exponentielle (utilisables en hors ligne)**
  - ➔ **Pour les autres problèmes non polynomiaux : on ne connaît que des solutions sous-optimales de complexité polynomiale**

Récapitulatif

- **Modèle de tâches : défini par les contraintes des tâches**
- **Pour chaque modèle de tâche, on a un ou plusieurs algorithmes d'ordonnancement.**
- **Mise en œuvre d'une application sur une machine cible**
  A faire HORS LIGNE :
  - ➔ **Vérifier que le problème est faisable (indépendamment de tout algorithme d'ordonnancement).**
  - ➔ **Choisir un algorithme d'ordonnancement.**
  - ➔ **Vérifier l'ordonnançabilité en utilisant les propriétés de l'algorithme choisi.**
  - ➔ **Si l'application est ordonnançable, alors l'implanter.**

Récapitulatif (suite)

- **Tout algorithme d'ordonnancement de tâche TR**

  ➔ **S'appuie/utilise un ordonnanceur de bas niveau (ou dispatcheur) qui s'exécute en ligne en même temps que les tâches d'application.**

  ➔ **Fait certains calculs hors ligne (eg. Affectation des priorités statiques)**

  ➔ **Effectue certains calculs en ligne (eg. Tri des files d'attente, calcul de la laxité…).**

Attention : si la vérification de l'ordonnaçabilité n'a pas été faite a priori, des fautes temporelles peuvent surgir en ligne et l'ordonnanceur n'est pas censé les détecter. L'ordonnanceur peut continuer à allouer le processeur à des tâches alors que leurs échéances sont ratées.

Chapitre 3

Ordonnancement de tâches indépendantes

1. Principaux algorithmes d'ordonnancement de tâches périodiques indépendantes

- **A priorités statiques**
  - **Rate Monotonic : fondé sur les périodes**
  - **Deadline Monotonic : fondé sur les délais critiques**
- **A priorités dynamiques**
  - **Earliest Deadline First (EDF) : fondé sur les échéances**
  - **Least Laxity First (LLF) : fondé sur la marge de manœuvre restante**

Rate Monotonic (Liu et Layland 1973)

Principe
Priorité = F(Période)
Plus petite période ⇒ Plus prioritaire

La priorité est inversement proportionnelle à la période

- **RM est optimal (pour $P_i = D_i$)**

  Condition d'ordonnançabilité (pour $P_i = D_i$) :

  - **CS (théorème de la limite d'utilisation - Lui 73)**

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

$CS(n) = n(2^{1/n} - 1)$
CS(2) = 0,828
CS(3) = 0,780
CS(4) = 0,757
CS(5) = 0,748
…
CS(n) → log 2 qd n → ∝

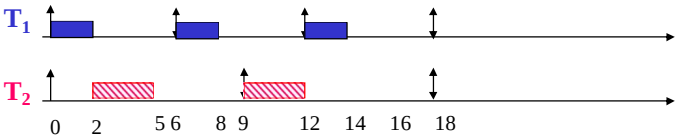
- **RM n'est pas optimal pour $P_i > D_i$**
- **RM n'est pas optimal pour l'ordonnancement non préemptif**

RM : exemple 1

$r_1 = r_2 = 0$ $\quad D_1 = P_1 = 6$ $\quad D_2 = P_2 = 9$

$C_1 = 2$ $\quad C_2 = 3$

$Prio_2$ < $Prio_1$



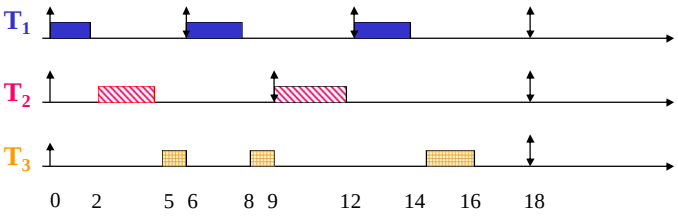
$U = 0,67 < 2(2^{1/2} - 1) = 0,83 \Rightarrow$ CS vérifiée

RM : exemple 2

$r_1 = r_2 = r_3 = 0$ $\quad D_1 = P_1 = 6$ $\quad D_2 = P_2 = 9$ $\quad D_3 = P_3 = 18$

$C_1 = 2$ $\quad C_2 = 3$ $\quad C_3 = 4$

$Prio_3$ < $Prio_2$ < $Prio_1$



$U = 0,89 > 3(2^{1/3} - 1) = 0,78 \Rightarrow$ CS non vérifiée

- **Condition nécessaire et suffisante**
  Théorème de Lehoczky, Sha et Ding 1987

Un ensemble de tâches (avec $D_i = P_i$) indépendantes est ordonnançable par RM ssi :

$$\forall i, 1 \le i \le n, \min_{(k,m) \in R_i} \left\{ \frac{1}{mP_k} \sum_{j=1}^{i} C_j \left\lceil \frac{mP_k}{P_j} \right\rceil \right\} \le 1$$

avec :

$$R_i = \left\{ (k,m) \middle| 1 \le k \le i, m = 1, \ldots, \left\lfloor \frac{P_i}{P_k} \right\rfloor \right\}$$

➔ Les indices des tâches son dans l'ordre croissant des périodes : $P_1 \le P_2 \le \ldots \le P_n$

Exemple d'utilisation de la CNS du théorème de Lehoczky et al. (1/3)

Exemple de configuration

Tâche	P_i	C_i	r_0
T1	5	2	0
T2	10	2	0
T3	20	4	0

$$\sum_{i=1}^{3} \frac{C_i}{P_i} \le 3(2^{\frac{1}{3}} - 1)$$

$$\sum_{i=1}^{3} \frac{C_i}{P_i} = \frac{2}{5} + \frac{2}{10} + \frac{4}{20} = 0.8 > 3(2^{\frac{1}{3}} - 1) = 0.779$$

CS de Liu non satisfaite

$$R_1 = \left\{ (k,m) \middle| 1 \le k \le 1, m = 1, \ldots, \left\lfloor \frac{P_1}{P_k} \right\rfloor \right\} = \{(1,1)\}$$

$$i = 1, \min_{(k,m) \in R_1} \left\{ \frac{1}{mP_1} \sum_{j=1}^{1} C_j \left\lceil \frac{mP_k}{P_j} \right\rceil \right\} \le 1$$

$$\frac{1}{P_1} C_1 \left\lceil \frac{P_1}{P_1} \right\rceil = \frac{C_1}{P_1} = \frac{2}{5} \le 1$$ ⇒ **T_1 est ordonnançable**

Exemple d'utilisation de la condition NS du Théorème de Lehoczky et al. (2/3)

$$i = 2, \min_{(k,m)\in R_2} \left\{ \frac{1}{mP_k} \sum_{j=1}^{2} C_j \left\lceil \frac{mP_k}{P_j} \right\rceil \right\} \le 1$$

$$(1,1) \Rightarrow \frac{1}{P_1} \sum_{j=1}^{2} C_j \left\lceil \frac{P_1}{P_j} \right\rceil = \frac{C_1}{P_1} \left\lceil \frac{P_1}{P_1} \right\rceil + \frac{C_2}{P_1} \left\lceil \frac{P_1}{P_2} \right\rceil = \frac{C_1}{P_1} + \frac{C_2}{P_1} = \frac{4}{5} \le 1$$ ⇒ **T_2 est ordonnançable**

$$(1,2) \Rightarrow \frac{1}{2P_1} \sum_{j=1}^{2} C_j \left\lceil \frac{2P_1}{P_j} \right\rceil = \frac{1}{2P_1} \left(C_1 \left\lceil \frac{2P_1}{P_1} \right\rceil + C_2 \left\lceil \frac{2P_1}{P_2} \right\rceil \right) = \frac{2C_1 + C_2}{2P_1} = \frac{6}{10} \le 1$$ ⇒ **T_2 est ordonnançable**

$$(2,1) \Rightarrow \frac{1}{P_2} \sum_{j=1}^{2} C_j \left\lceil \frac{P_2}{P_j} \right\rceil = \frac{1}{P_2} \left(C_1 \left\lceil \frac{P_2}{P_1} \right\rceil + C_2 \left\lceil \frac{P_2}{P_2} \right\rceil \right) = \frac{2C_1 + C_2}{P_2} = \frac{6}{10} \le 1$$ ⇒ **T_2 est ordonnançable**

Il suffit de trouver un couple (k,m) qui vérifie la relation ≤ 1
Ici on a traité les trois couples (k,m), juste pour mieux comprendre l'application du théorème

Exemple d'utilisation de la condition NS du Théorème de Lehoczky et al. (3/3)

$$R_3 = \left\{ (k,m) \middle| 1 \le k \le 3, m = 1, \ldots, \left\lfloor \frac{P_3}{P_k} \right\rfloor \right\} = \{(1,1),(1,2),(1,3),(1,4),(2,1),(2,2),(3,1)\}$$

$$(1,1) \Rightarrow \frac{1}{P_1} \sum_{j=1}^{3} C_j \left\lceil \frac{P_1}{P_j} \right\rceil = \frac{C_1}{P_1} \left\lceil \frac{P_1}{P_1} \right\rceil + \frac{C_2}{P_1} \left\lceil \frac{P_1}{P_2} \right\rceil + \frac{C_3}{P_1} \left\lceil \frac{P_1}{P_3} \right\rceil = \frac{C_1}{P_1} + \frac{C_2}{P_1} + \frac{C_3}{P_1} = \frac{8}{5} \ge 1$$ ⇒ **On ne peut pas conclure encore**

$$(1,2) \Rightarrow \frac{1}{2P_1} \sum_{j=1}^{3} C_j \left\lceil \frac{2P_1}{P_j} \right\rceil = \frac{C_1}{2P_1} \left\lceil \frac{2P_1}{P_1} \right\rceil + \frac{C_2}{2P_1} \left\lceil \frac{2P_1}{P_2} \right\rceil + \frac{C_3}{2P_1} \left\lceil \frac{2P_1}{P_3} \right\rceil = \frac{2C_1}{2P_1} + \frac{C_2}{2P_1} + \frac{C_3}{2P_1} = \frac{10}{10} \le 1$$ ⇒ **T_3 est ordonnançable**

On pourrait faire aussi le test pour les 5 couples (k,m) qui restent mais c'est inutile.

Deadline Monotonic (Leung et Whitehead 1982)

Principe
Priorité = F(Délai critique)
Plus petit délai critique ⇒ Plus prioritaire

La priorité est inversement proportionnelle au délai critique

- **DM est optimal pour l'ordonnancement préemptif si** $P_i \geq D_i$

Condition d'ordonnançabilité : (Leung et Whitehead 1982)

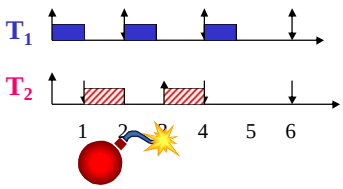
- CS : $\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n(2^{\frac{1}{n}} - 1)$

- **DM est optimal pour le non-préemptif si** $\forall i, \forall k, C_i \leq C_k \Rightarrow D_i \leq D_k$
- **DM n'est pas optimal pour l'ordonnancement non préemptif en général**

RM et DM : exemple

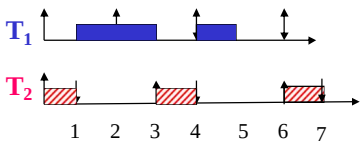
$r_1 = r_2 = 0$ $\quad D_1 = P_1 = 2$ $\quad D_2 = 1\ P_2 = 3$

$C_1 = 1$ $\quad C_2 = 1$



RM

$Prio_2$ < $Prio_1$



DM

$Prio_1$ < $Prio_2$

- **Condition nécessaire et suffisante (condition du temps de réponse)**
  Théorème de Audsley et al. 1993

Un ensemble de tâches (avec $D_i \leq P_i$) indépendantes et indexées selon leurs échéances (i.e. : $i > j \Rightarrow Di \geq Dj$) est ordonnançable par DM ssi :

$$\forall i, 1 \leq i \leq n, R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{P_j} \right\rceil C_j \leq D_i$$

$R_1 = C_1$

***R_i* (i≥2) se calcule de manière itérative comme suit :**

- On fixe R_i^0 à 0
- On calcule R_i^1 à partir de R_i^0
- ...
- On calcule R_i^n à partir de R_i^{n-1}
- On s'arrête quand R_i^{n+1} est égal à R_i^n

Exemple d'utilisation de la condition NS du Théorème de Audsley

Exemple de configuration

Tâche	P_i	C_i	D_i
T1	12	3	8
T2	20	6	10

$$\sum_{i=1}^{2} \frac{C_i}{D_i} \leq 2(2^{\frac{1}{2}} - 1)$$

$$\sum_{i=1}^{2} \frac{C_i}{D_i} = \frac{3}{8} + \frac{6}{10} = 0.975 > 2(2^{\frac{1}{2}} - 1) = 0.83$$

CS de Leung et Whitehead 1982 non satisfaite

$$i = 1 \Rightarrow R_1 = C_1 = 3 \leq D_i = 8$$

$$i = 2 \Rightarrow R_2 = C_2 + \sum_{j=1}^{1} \left\lceil \frac{R_2}{P_j} \right\rceil C_j = C_2 + C_1 \left\lceil \frac{R_2}{P_1} \right\rceil \leq D_2$$

$$R_2^0 = 0$$

$$R_2^1 = C_2 + C_1 \left\lceil \frac{R_2^0}{P_1} \right\rceil = C_2 = 6$$

$$R_2^2 = C_2 + C_1 \left\lceil \frac{R_2^1}{P_1} \right\rceil = 6 + 3 \left\lceil \frac{6}{12} \right\rceil = 9$$

$$R_2^3 = C_2 + C_1 \left\lceil \frac{R_2^2}{P_1} \right\rceil = 6 + 3 \left\lceil \frac{9}{12} \right\rceil = 9 \leq 10$$

T_1 et T2 sont ordonnançables

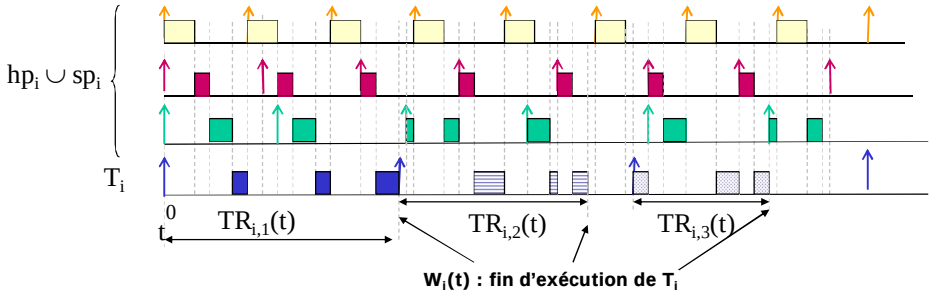
Calcul du temps de réponse maximum de tâche périodique à priorité fixe

Cas préemptif (cas synchrone i.e. $\forall k, r_k = 0$)

Cas $D_i \leq P_i$

Une tâche plus prioritaire peut retarder T_i jusqu'à la fin de son exécution

hp_i : ensemble de tâches plus prioritaires que la tâche T_i
sp_i : ensemble de tâches de même priorité que la tâche T_i
$Tr_i(t)$: temps de réponse de la tâche T_i devenue activable à t



Le pire cas apparaît pendant la première période d'activité du processeur

$$TR_i = W_i(0) = C_i + \sum_{T_j \in hp_i \cup sp_i} \left\lceil \frac{W_i(0)}{P_j} \right\rceil C_j$$

Earliest Deadline First (Liu et Layland 1973)

Principe
Echéance la plus proche ⇒ Plus prioritaire

La priorité est inversement proportionnelle à l 'échéance

EDF est optimal pour les systèmes de tâches indépendantes (pour $P_i = D_i$)

CNS pour les systèmes à échéance sur requête ($P_i = D_i$) **:**

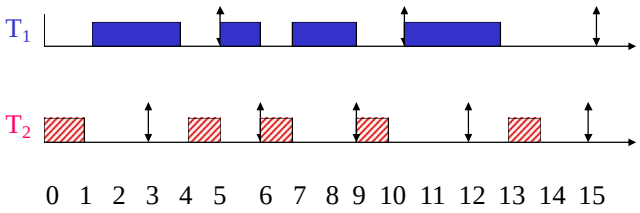
$U \leq 1$ **(Liu et Layland 1973)**

CS pour les systèmes avec $P_i \neq D_i$) **:** $\sum_{i=1}^{n} \frac{C_i}{\min(D_i, P_i)} \leq 1$

EDF : exemple

$r_1 = r_2 = 0$ $\quad D_1 = P_1 = 5$ $\quad D_2 = P_2 = 3$

$C_1 = 3$ $\quad C_2 = 1$



0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

U = 3/5 + 1/3 = 14/15 < 1 $\Rightarrow$ le système est ordonnançable par EDF

- **Condition nécessaire et suffisante (condition de la demande de processeur) Théorème de Jeffay et Stone 1993**

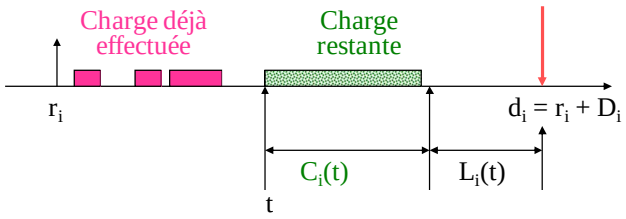
Un ensemble de tâches (avec $D_i \leq P_i$) est ordonnançable par EDF ssi :

$$\forall t > 0, t \geq \sum_{i=1}^{n} \left(\left\lfloor \frac{t - D_i}{P_i} \right\rfloor + 1 \right) \times C_i$$

Least Laxity First (Sorenson 1974)

Principe :
Laxité(t) = Li(t) = $r_i + D_i - t - C_i(t)$
Laxité plus petite ⇒ Prioritaire plus élevée

La priorité est inversement proportionnelle à la laxité



Même capacité d'ordonnancement que EDF

Plus équitable que EDF (utile en cas d'allocation équitable)

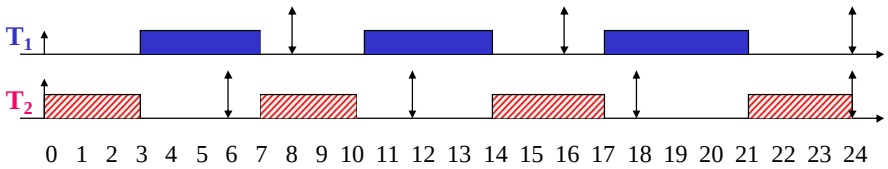
Défaut : **Nombre élevé de changements de contexte**
Recalcul des priorités toutes les unités de temps

EDF et LLF : exemple 1 (1/2)

$r_1 = r_2 = 0$

$D_1 = P_1 = 8$ $C_1 = 4$

$D_2 = P_2 = 6$ $C_2 = 3$

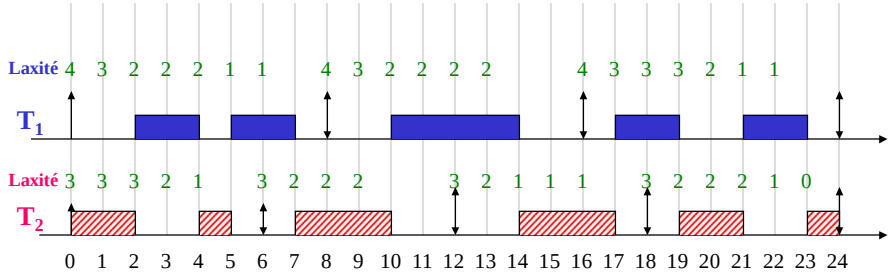


Avec EDF : 6 changements de contexte

EDF et LLF : exemple 1 (2/2)

$r_1 = r_2 = 0$ $\quad D_1 = P_1 = 8$ $\quad D_2 = P_2 = 6$

$C_1 = 4$ $\quad C_2 = 3$

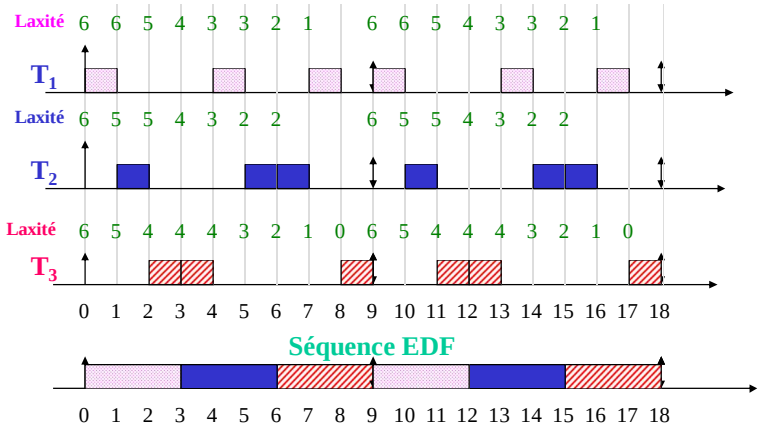


Avec LLF : 10 changements de contexte

EDF et LLF : exemple 2

$r_1 = 0, D_1 = 9, P_1 = 9, C_1 = 3$ $\quad r_2 = 0, D_2 = 9, P_2 = 9, C_2 = 3$

$r_3 = 0, D_3 = 9, P_3 = 9, C_3 = 3$



2. Ordonnancement de tâches apériodiques indépendantes

- Cas des tâches apériodiques avec contraintes de délai

Alarme coupure courant

$Tap_i(r_i, C_i, D_i)$

➔ Tâches critiques : transformation (à la conception) de tâches apériodiques en tâches périodiques. Dans ce cas l'intervalle minimum doit être connu et sera utilisé comme période. On parle ici de tâche sporadique.
Ou en utilisant un serveur

➔ Tâches non critiques : acceptation dans les temps creux d'une séquence rigide de tâches (utilisation de test d'acceptation)

- Cas des tâches apériodiques sans échéance

Transfert d'image

$Tap_i(r_i, C_i)$ ⇒ Pas d'échéance ; Optimiser le temps de réponse

➔ Traitement en arrière plan (background processing)
les tâches apériodiques sont traitées pendant les temps d'oisiveté du processeur (quand il n'y a pas de tâches périodiques à exécuter).

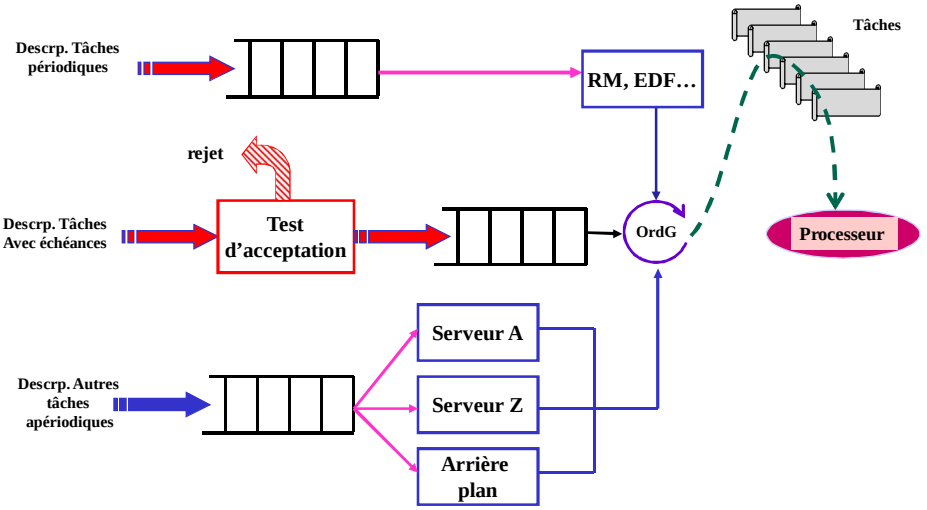
➔ Utilisation d'un serveur de tâches apériodiques

➔ Vol de temps creux

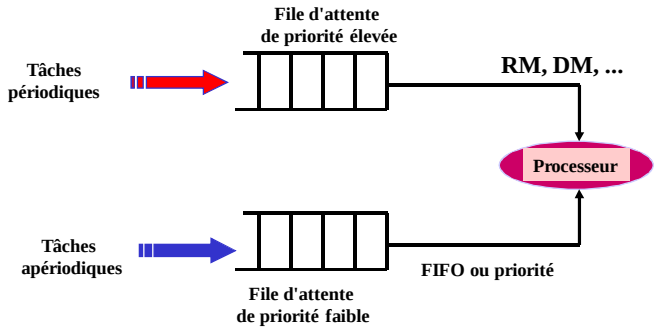
Types de serveurs de tâches apériodiques

- serveur à scrutation
- serveur ajournable
- serveur sporadique
- serveur à échange de priorité
- serveur Earliest Deadline Last (EDL)
- serveur à largeur de bande maximale
- autres

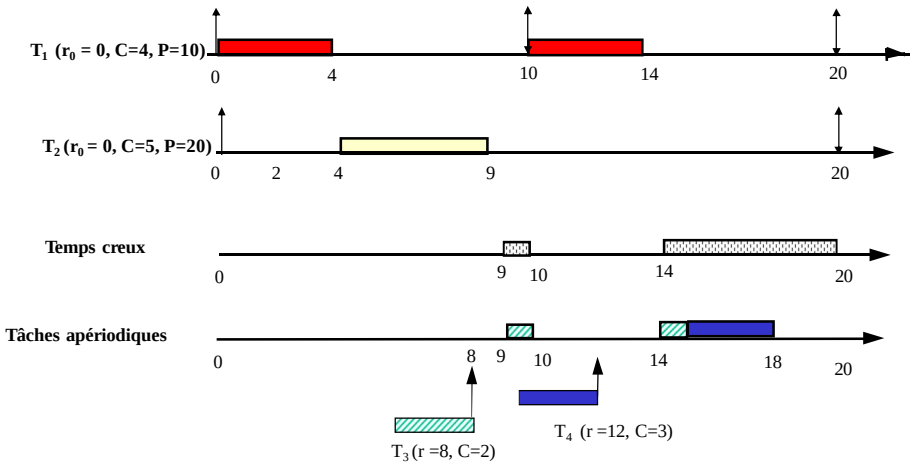
Différentes stratégies pour le traitement de tâches apériodiques



Traitement en arrière plan



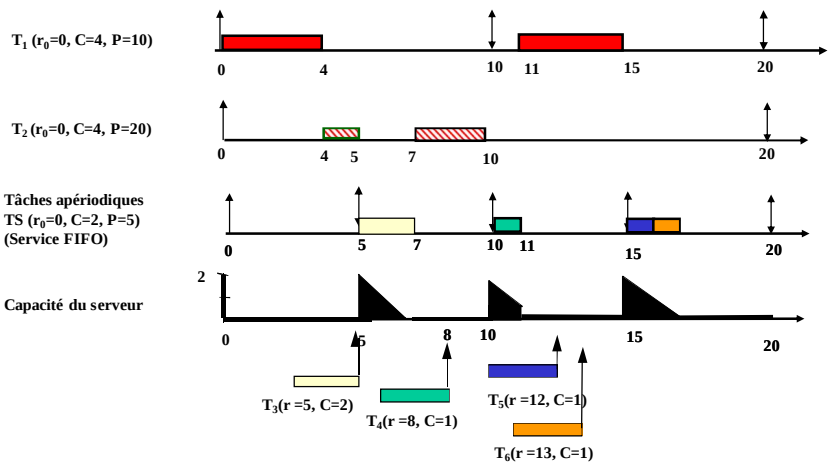
Traitement en arrière plan : exemple



Serveur à scrutation (Polling server)
(Lehoczky, Sha et Strosnider 1987)

- **Une tâche périodique spécifique, appelée serveur, est dédiée aux traitements des requêtes apériodiques.**
- **Le serveur à scrutation traite les tâches apériodiques qu'il trouve en attente lors de son activation.**
- **S'il n'y a pas de tâches en attente, le serveur se suspend et attend sa prochaine période d 'activation. Il perd sa capacité de service.**
- **Un ou plusieurs serveurs à scrutation peuvent être utilisés selon les besoins.**

Serveur à scrutation avec RM : exemple



Test de garantie de tâches apériodiques en cas d'utilisation de Serveur à scrutation

- P_s : période du serveur C_s : capacité du serveur

Cas d'une seule tâche apériodique

- Tâche apériodique définie par:
  r_a (instant d'arrivée), C_a (durée d'exécution) et D_a (délai)
- Une tâche apériodique (quand elle est seule) peut attendre P_s au plus avant de s'exécuter
- Si $C_a \leq C_s$ alors garantie si $2P_s \leq D_a$
- Condition suffisante : Si C_a est quelconque alors garantie si $P_s\left(\left\lceil \frac{C_a}{C_s} \right\rceil + 1\right) \leq D_a$

- Condition nécessaire et suffisante quand le serveur a la plus haute priorité parmi toutes les tâches périodiques

$$\left(\left\lfloor \frac{C_a}{C_s} \right\rfloor + \left\lceil \frac{r_a}{P_s} \right\rceil\right) P_s + \left(C_a - \left\lfloor \frac{C_a}{C_s} \right\rfloor C_s\right) \leq r_a + D_a$$

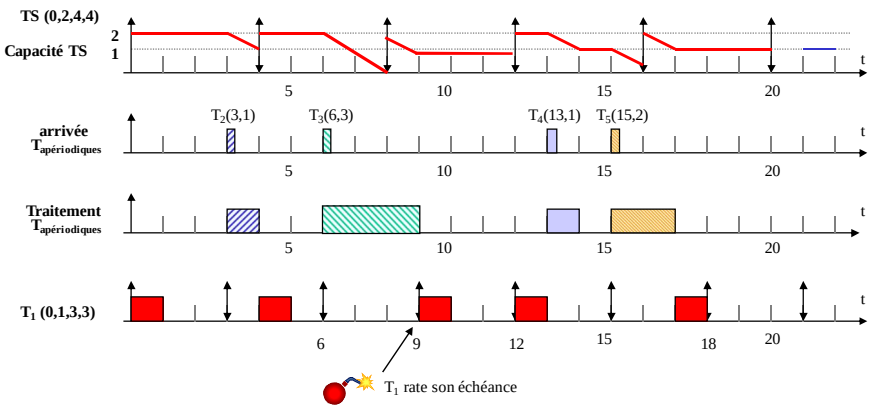
Cas de k tâches apériodiques ordonnées selon leur délai

- Des conditions (CS et CNS) complexes existent

Serveur ajournable (deferrable server)
(Lehoczky, Sha et Strosnider 1987)

- **Une tâche périodique spécifique, appelée serveur, est dédiée aux traitements des requêtes apériodiques.**
- **Le serveur ajournable traite les tâches apériodiques qu'il trouve en attente lors de son activation.**
- **S'il n'y a pas de tâches en attente, le serveur se suspend et attend la prochaine période d'activation. Il conserve sa capacité de service.**
- **Si une requête apériodique arrive alors que le serveur ajournable est suspendu, elle est exécutée si la capacité du serveur n'est pas nulle (d'où violation du principe de RM)**
- **En début de période, la capacité du serveur est remise dans son état initial indépendamment de la capacité précédemment consommée.**
- **Un ou plusieurs serveurs ajournables peuvent être utilisés selon les besoins.**

Serveur ajournable avec RM : exemple 1



Test de garantie de tâches apériodiques en cas d'utilisation de Serveur ajournable

- P_s : période du serveur $C_s(t)$: capacité du serveur à l'instant t

Cas d'une seule tâche apériodique et serveur avec la plus haute priorité

- Tâche apériodique définie par r_a (instant d'arrivée), C_a (durée d'exécution) et D_a (délai)
- Une tâche apériodique peut s'exécuter dès qu'elle arrive si $C_s(r_a) > 0$
- Condition ~~nécessaire et~~ suffisante :

$$\begin{cases} r_a + C_a \le r_a + D_a \quad Si \quad C_a \le C_s(r_a) \\ \left(\left\lfloor \frac{C_a - \Delta_a}{C_s} \right\rfloor + \left\lceil \frac{r_a}{P_s} \right\rceil\right) P_s + C_a - \Delta_a - \left\lfloor \frac{C_a - \Delta_a}{C_s} \right\rfloor C_s \le r_a + D_a \quad Sinon \end{cases}$$

$$\Delta_a = \min\left[C_s(r_a), \left(\left\lceil \frac{r_a}{P_s} \right\rceil P_s - r_a \right) \right]$$

Serveur sporadique (sporadic server)

(Sprunt, Sha et Lehoczky 1989)

- Le serveur sporadique permet d'améliorer les temps de réponse des tâches apériodiques, sans diminuer le taux d'utilisation du processeur des tâches périodiques.
- Le serveur sporadique ne réapprovisionne pas sa capacité de manière périodique comme les serveurs à scrutation et ajournable, mais seulement quand elle a été consommée par des tâches apériodiques.
- Efficacité du concept de serveur dynamique
  - Utile s'il existe plusieurs serveurs qui fonctionnent en parallèle dans le système. On donne plus de capacité à celui qui en a besoin.
  - Encore plus utile si on peut adapter dynamiquement la capacité maximale du SS.
- Il existe plusieurs formes de serveurs sporadiques, pour priorités fixes ou dynamiques (les serveurs sporadiques se distinguent par la manière de consommer et réapprovisionner la capacité de service)

Fonctionnement d'un serveur sporadique simple (avec priorités fixes)

PR_{exe} : priorité de la tâche en cours d'exécution.

PR_s : priorité du serveur sporadique.

P_s : période du serveur sporadique. C_s : capacité maximale du serveur sporadique.

***RT* : instant où aura lieu le réapprovisionnement de la capacité du SS.**

***RA* :quantité de réapprovisionnement qui sera rajoutée à la capacité du SS.**

Actif : le SS est dit actif quand $PR_{exe} \geq PR_s$. (SS actif ne signifie pas qu'il consomme du temps CPU)

Inactif : le SS est dit inactif quand $PR_{exe} < PR_s$.

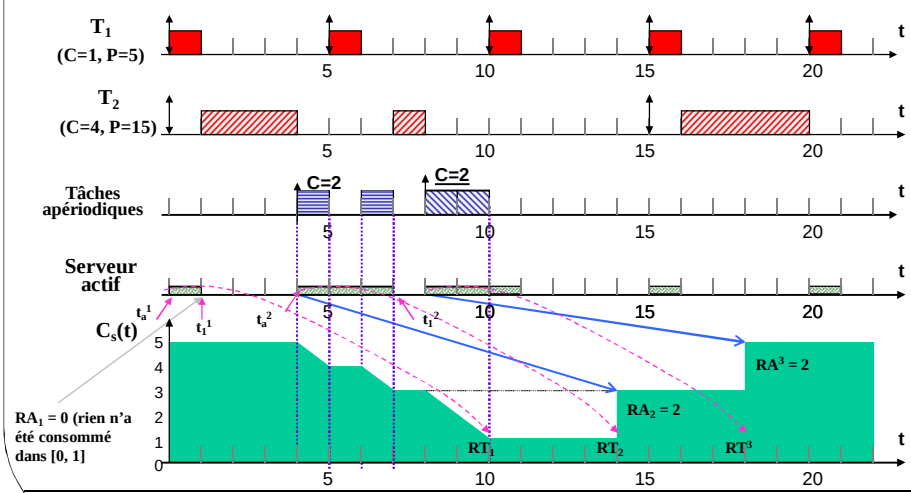
Quand une tâche apériodique arrive, elle s'exécute si la capacité du serveur n'est pas nulle et si la priorité du serveur le permet (les tâches apériodiques s'exécutent avec le même niveau de priorité que le serveur sporadique).

Règle de réapprovisionnement de la capacité du serveur sporadique

- **A l'initialisation, la capacité du SS est approvisionnée avec la valeur maximale C_s.**
- **L'instant de réapprovisionnement RT est fixé dès le moment où le SS devient activable et $C_s > 0$ (soit t_a ce moment). La valeur de RT est fixé à $t_a + P_s$.**
- **La quantité de réapprovisionnement, RA, à effectuer à l'instant RT est fixée au moment où le SS devient inactif ou quand sa capacité est épuisée (soit t_1 ce moment). La valeur de RA est fixée à une quantité égale à celle consommée durant l'intervalle $[t_a, t_1]$.**

Serveur sporadique avec RM : exemple (1/2)

T_s ($r_0 = 0$, $C_s = 5$, $P_s = 10$, $D_s = 10$)



Serveur sporadique avec RM : exemple (2/2)

- A $t = 0$, T_1 (tâche la plus prioritaire) est activée et le SS (serveur sporadique) devient actif. Comme $C_s > 0$, RT_1 est fixé à $t + P_s = 10$. ($t_a^1 = 0$)
- A $t = 1$, T_1 se termine, le SS devient inactif, car il n'a pas de tâche apériodique à servir. RA_1 (pour $t_1^1 = 1$) est égale à 0 et aucun réapprovisionnement n'est effectué à l'instant RT_1, car aucune quantité de la capacité du SS n'a été consommée dans l'intervalle [0, 1].
- A $t = 4$, une tâche apériodique, T_3, arrive. Comme $C_s > 0$, le SS devient actif et T_3 s'exécute. Par conséquent un réapprovisionnement est prévu à $RT_2 = t + P_s = 14$. ($t_a^2 = 4$)
- A $t = 5$, la tâche apériodique T_3 est préemptée par T_1. Le SS reste actif pendant la préemption.
- A $t = 6$, T_1 se termine et T_3 reprend son exécution et se termine à $t = 7$.
- A $t = 7$, le SS devient inactif. La quantité de réapprovisionnement à faire à RT_2 est égale à la capacité consommée pendant l'intervalle [4, 7], c'est-à-dire, $RA_2 = 2$.
- A $t = 8$, le SS devient actif pour servir une nouvelle tâche apériodique, T_4, arrivée à $t = 8$. Un nouveau réapprovisionnement est prévu à $RT_3 = t + P_s = 18$. ($t_a^3 = 8$)
- A $t = 10$, T_4 se termine, mais le SS reste actif car $PR_{exe} \geq PR_s$.
- A $t = 11$, le SS devient inactif et la quantité de réapprovisionnement à faire à l'instant RT_3 est fixée à $RA_3 = 2$ (2 étant la capacité consommée dans l'intervalle [8, 11]).

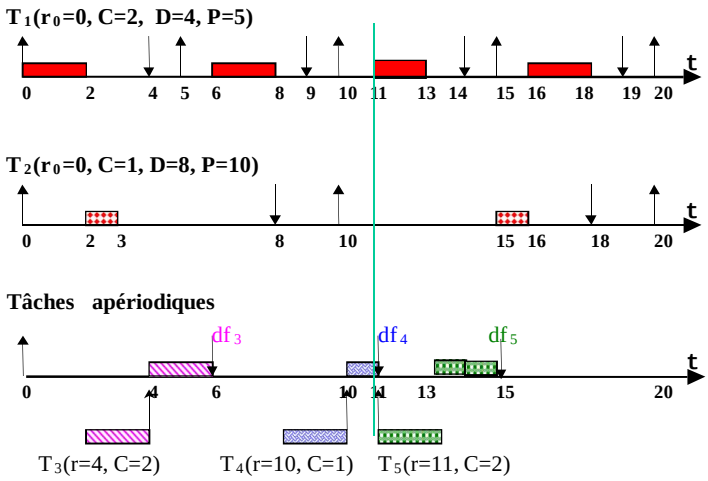
Algorithmes à vol de temps creux
(Slack-stealing algorithms) - (Lehoczky et Ramos-Thuel 1992)

- Pour les systèmes dirigés par les délais (ex. EDF ou LLF)
- Aucune tâche périodique n'est dédiée au traitement des tâches apériodiques.
- Principe de fonctionnement :

  lorsqu'une tâche apériodique est activée, le système recule au maximum l'instant de déclenchement des tâches périodiques pour autant qu'elles continuent à respecter leurs contraintes temporelles.

  Quand EDF est utilisé, on calcule pour chaque tâche apériodique une échéance fictive df qui est celle qu'il faut lui associer pour qu'elle soit exécutée avec un temps de réponse minimal. df correspond alors à la première date pour laquelle le temps d'exécution de la tâche est égal au temps total d'inactivité du processeur quand l'ordonnancement respecte les échéances de toutes les tâches périodiques et des tâches apériodiques précédemment déclenchées et non encore achevées.

Slack-stealing avec EDF : exemple (1/2)



Slack-stealing : exemple (2/2)

- **À *t=4*, la tâche apériodique $T_3(r=4, C=2)$ est activée.**

- **La deuxième requête de la tâche T_1 se réveillant à *t=5* a une laxité de 2 unités de temps ; elle peut donc reculer le début de son exécution jusqu'à *t=6*.**

- **La tâche T_3 peut donc s'exécuter immédiatement entre *t=4* et *t=6* et sa date d'échéance fictive df_3 est alors égale à *6*.**

- **De la même façon, la troisième requête de la tâche T_1 peut reculer le début de son exécution jusqu'à *t=11* de façon à laisser la tâche T_4 s'exécuter immédiatement suite à leur réveil. La tâche T_4 a donc une échéance fictive fixée à $df_4 = 11$.**

- **La tâche T_5 obtient une échéance fictive $df_5 = 15$ et s'exécute après la fin de la troisième requête de T_1 (c'est-à-dire dans l'intervalle [13,15]).**

Comparaison des algorithmes de traitement de tâches apériodiques

Technique de traitement	Performance (opt. du temps de rép.)	Complexité de calcul	Complexité implémentation
Traitement en arrière plan	↘	↗	↗
Serveur à scrutation	↘	↗	↗
Serveur ajournable	→	↗	↗
Serveur sporadique	→	→	→
Slack-Stealer	↗	↘	↘

↗ : excellent → : bon ↘ : mauvais

Théorèmes de Tia, Liu et Shankar 1995

1. Il n'existe pas d'algorithme valide qui minimise le temps de réponse pour chaque tâche apériodique à contraintes relatives, pour tout ensemble de tâches périodiques ordonnées par priorités fixes et tout ensemble de tâches apériodiques ordonnées selon une discipline donnée.

2. Il n'existe pas d'algorithme (en ligne) valide qui minimise le temps de réponse moyen des tâches apériodiques à contraintes relatives, pour tout ensemble de tâches périodiques ordonnées par priorités fixes et tout ensemble de tâches apériodiques ordonnées selon une discipline donnée.

Chapitre 4

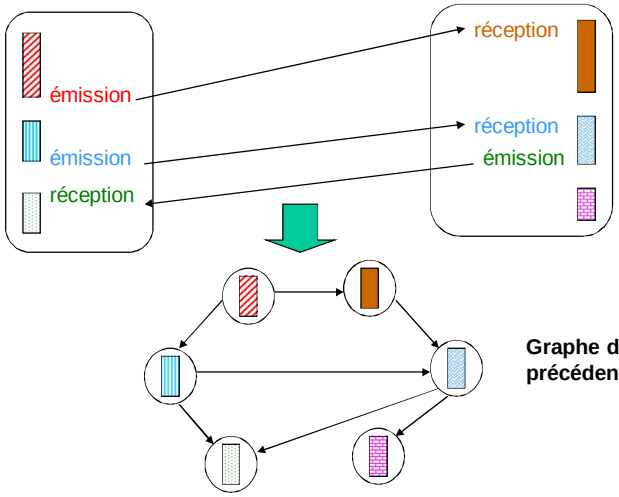
Ordonnancement de tâches dépendantes

1. Prise en compte des contraintes de précédence

- Prise en compte de la communication
- Synchronisation

- **Principe** : Se ramener à l'ordonnancement de tâches indépendantes
- **Méthode** : Ajustement des dates de réveil et échéances
  Utilisation d'un algorithme classique : RM, DM ou EDF
  Égalité des priorités ⇒ utilisation du graphe de précédence

$T_1 \longrightarrow T_2 \Rightarrow Prio_1 > Prio_2$



2 tâches de même période

Graphe de précédence

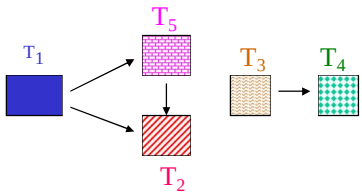
- **Résultat** **Le système ajusté est ordonnançable par l'algorithme choisi**
  ⟺ **Le système initial est ordonnançable**

Précédence et RM

Règles d'ajustement : $r^*_i = \max_k\{r_i, r^*_k, T_k \rightarrow T_i\}$

$Prio^*_i = \max\{Pro_i, Prio^*_k + a, T_i \rightarrow T_k\}$, $a > 0$

$r_1 = 1$	$r_2 = 0$	$r_3 = 0$	$r_4 = 0$	$r_5 = 0$
$D_1 = P_1 = 8$	$D_2 = P_2 = 8$	$D_3 = P_3 = 12$	$D_4 = P_4 = 12$	$D_5 = P_5 = 8$
$C_1 = 1$	$C_2 = 2$	$C_3 = 2$	$C_4 = 2$	$C_5 = 2$



$r_1^* = 1$	$r_2^* = 1$	$r_3^* = 0$	$r_4^* = 0$	$r_5^* = 1$

$Prio_4 < Prio_3 < Prio_2 < Prio_5 < Prio_1$

Exemple RM et précédence

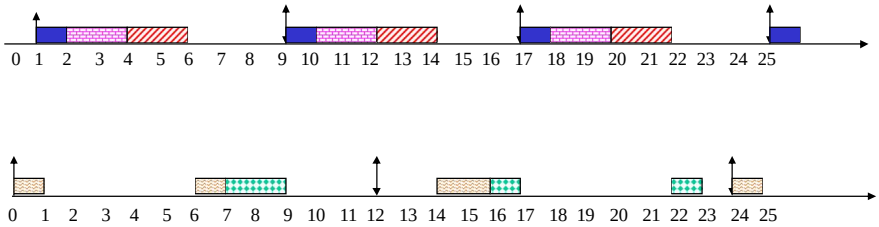
$r_1^* = 1$ $D_1 = P_1 = 8$ $C_1 = 1$

$r_2^* = 1$ $D_2 = P_2 = 8$ $C_2 = 2$

$r_3 = 0$ $D_3 = P_3 = 12$ $C_3 = 2$

$r_4 = 0$ $D_4 = P_4 = 12$ $C_4 = 2$

$r_5^* = 1$ $D_5 = P_5 = 8$ $C_5 = 2$

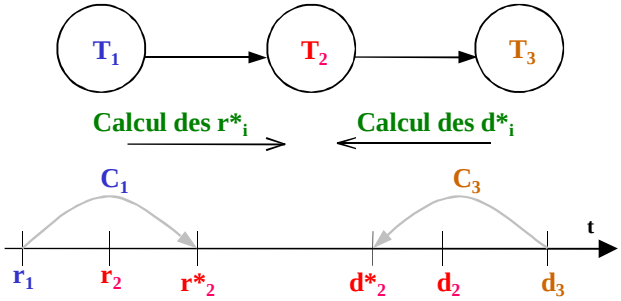


$Prio_4 < Prio_3 < Prio_2 < Prio_5 < Prio_1$

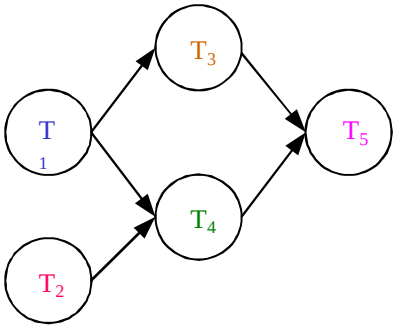
Précédence et EDF

Règle d'ajustement : $r^*_i = \underset{k}{Max}\{r_i, r^*_k + C_k, T_k \rightarrow T_i\}$

$d^*_i = \underset{k}{Min}\{d_i, d^*_k - C_k, T_i \rightarrow T_k\}$



Exemple EDF et précédence



	Paramètres des tâches			Paramètres transformés	
Tâche	r_i	C_i	d_i	r^*_i	d^*_i
T_1	0	1	5	0	3
T_2	5	2	7	5	7
T_3	0	2	5	1	5
T_4	0	1	10	7	9
T_5	0	3	12	8	12

2. Prise en compte des contraintes de partage de ressources

Fichiers, BD

Utilisation de ressources critiques

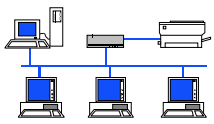
- **Mono ou multi exemplaires**
- **Mode lecture/écriture**
- **Demande de 1 ou n ressources**

Ressource physique

E/S

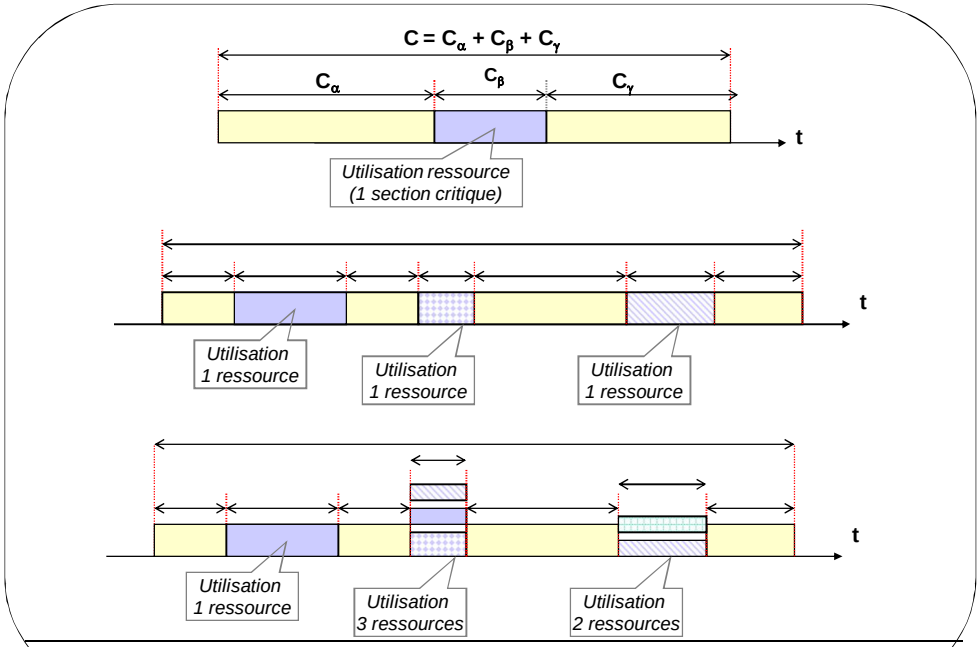
Difficultés :

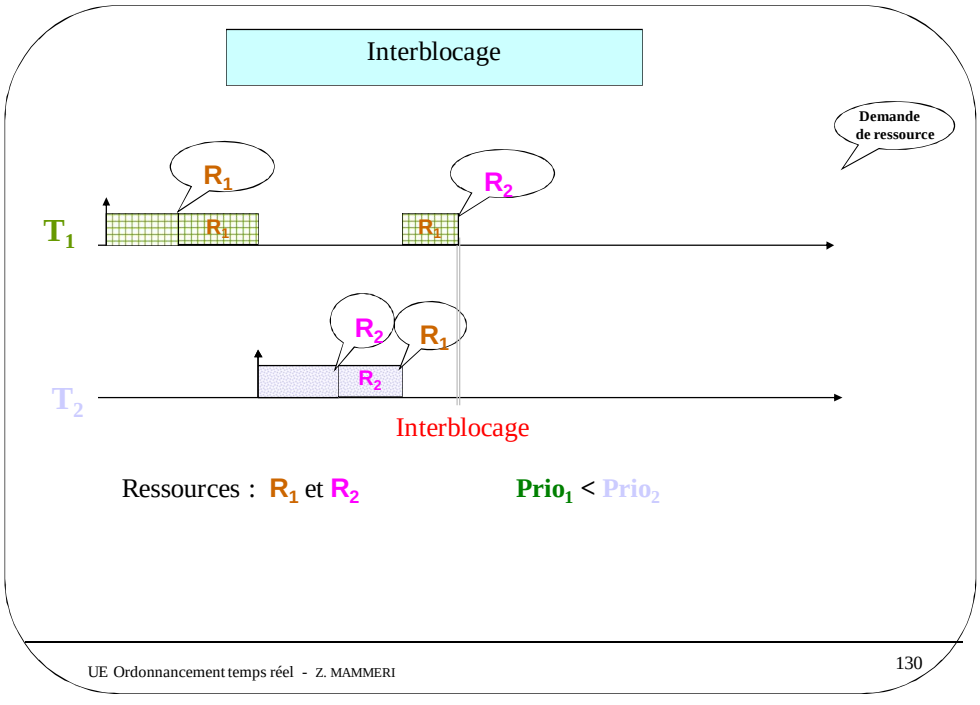
- **Interblocages**
- **Inversion de priorités**
- **Instabilité des algorithmes**



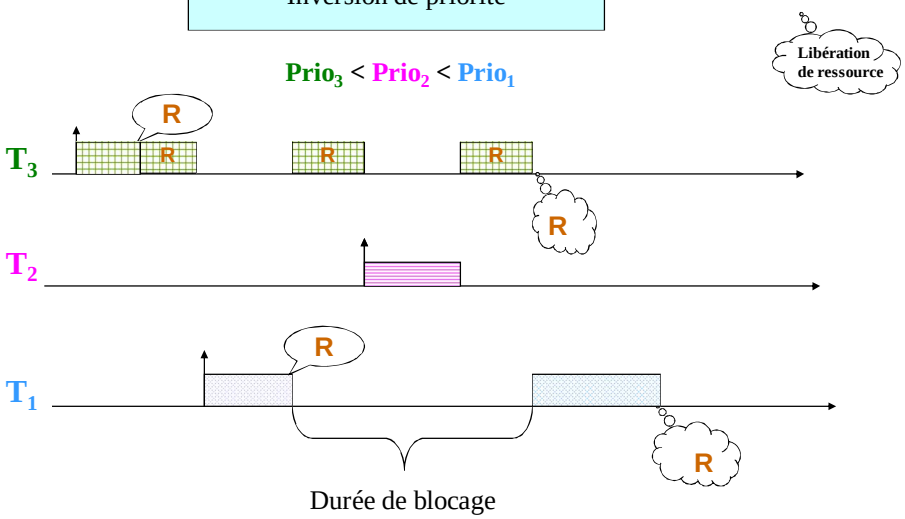
Réseau

Ordonnancement : NP-difficile, pas d'algo optimal [Mok 1983]


C = Cα + Cβ + Cγ
Cα
Cβ
Cγ
t
Utilisation ressource
(1 section critique)
t
Utilisation
1 ressource
Utilisation
1 ressource
Utilisation
1 ressource
t
Utilisation
1 ressource
Utilisation
3 ressources
Utilisation
2 ressources


Interblocage
Demande
de ressource
R1
R2
T1
R1
R1
R2
R1
T2
R2
Interblocage
Ressources : R1 et R2
Prio1 < Prio2

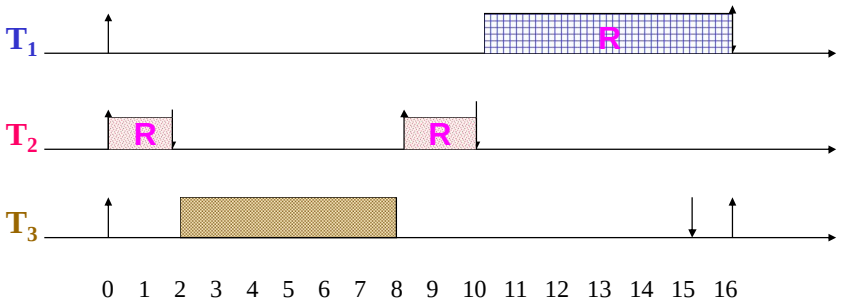
Inversion de priorité



Instabilité de EDF (1/2)

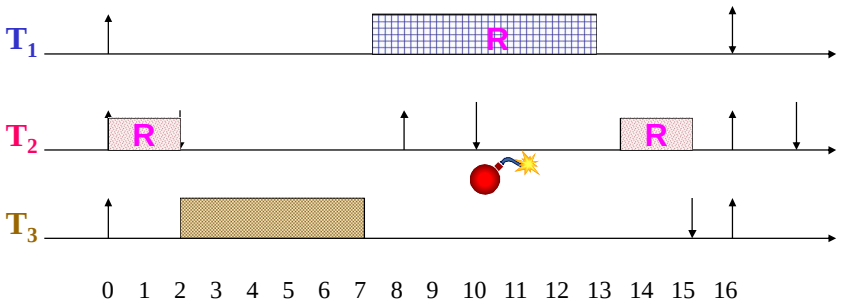
$r_1 = r_2 = r_3 = 0$ $D_1 = P_1 = 16$ $D_2 = 2\ P_2 = 8$ $D_3 = 15\ P_3 = 16$

$C_1 = 6$ $C_2 = 2$ $\mathbf{C_3 = 6}$



Instabilité de EDF (2/2)

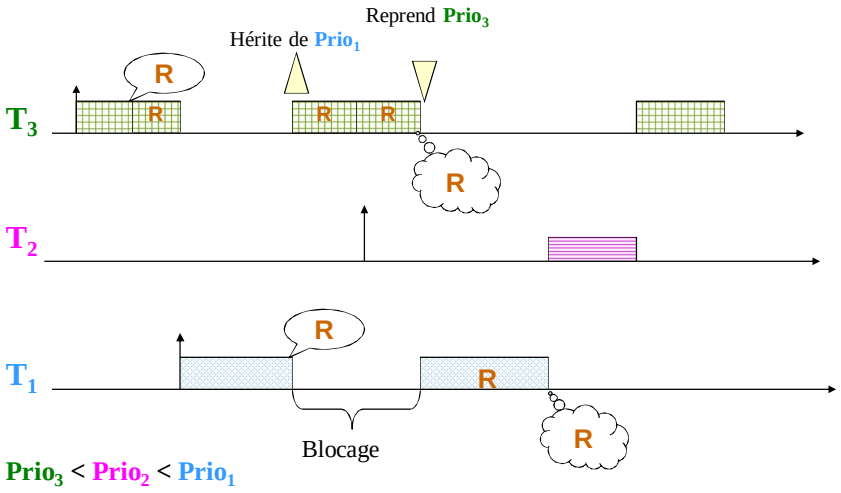
$r_1 = r_2 = r_3 = 0$ $D_1 = P_1 = 16$ $C_1 = 6$ $D_2 = 2\ P_2 = 8$ $C_2 = 2$ $D_3 = 15\ P_3 = 16$ $C_3 = 5$



Protocole à priorité héritée : PIP
(Priority Inheritance Protocol)
(Sha, Rajkumar et Lehoczky 1990)

- **Principe**
  - ➔ La tâche T_i qui possède une ressource critique prend une priorité égale au Max{$Prio_i$, $Prio_k$ | T_k en attente de la section critique}.
  - ➔ Quand T_i sort de la SC, elle reprend sa priorité initiale.
- **Utilisable pour les ressources mono exemplaires**

Protocole à priorité héritée PIP



Propriétés de PIP

- **Durée de blocage maxi de n'importe quelle tâche ordonnancée avec PIP ≤ Min(nb_tâches, nb_ressources) * pire_durée_SC**

- **Pour une tâche *i* et une seule ressource *R*:**

$$B_i = \max_{\forall k \notin hp(i)} (SC_k(R)) + \sum_{R} \max_{\forall j \in hp(i)} (durée\_SC_j(R))$$

hp(i) : ensemble de tâches plus prioritaires que τ_i. *R* : toute ressource utilisée par une tâche ∈hp(i).

- **Condition suffisante pour RM**

$$\max_{i=1,..,n}\left(\frac{B_i}{P_i}\right) + \sum_{k=1}^{n} \frac{C_k}{P_k} \le n(2^{1/n} - 1)$$

- **Condition suffisante pour EDF**

$$\forall i = 1,2,...,n \Rightarrow \left(\frac{C_i + B_i}{D_i}\right) + \sum_{k=1}^{k=i-1} \frac{C_k}{D_k} \le 1$$

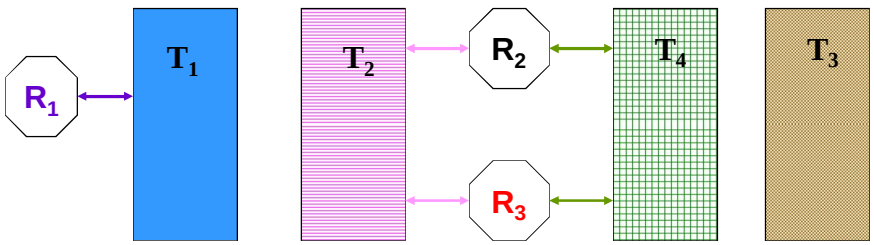
B_i : durée maxi de blocage de la tâche T_i en attente de SC.

- **PIP ne gère pas les interblocages**

Protocole à priorité plafond : PCP (Priority Ceiling Protocol)
(Sha, Rajkumar et Lehoczky 1990)

- **Principe**
  - Chaque ressource R_x possède une priorité plafond égale à $Max\{Prio_k \mid T_k$ pouvant accéder à $R_x\}$.
  - On note Π_s le maximum des priorités plafonds des ressources en cours d'utilisation par des tâches. Π_s est égal à une valeur inférieure à la priorité la plus basse quand aucune ressource n'est utilisée.
  - Quand une tâche T_i veut accéder à une ressource R_x qui est libre :
    - Cas 1 : si $Prio_i$ est supérieur à Π_s, alors T_i obtient R_x et Π_s est mis à jour.
    - Cas 2 : si $Prio_i$ n'est pas supérieur à Π_s, alors T_i obtient R_x seulement si T_i est la tâche qui détient la (ou les) ressource(s) dont le plafond est égal à Π_s.
    - Autres cas : T_i est bloquée.
  - Une tâche T_i qui est en SC prend la priorité $Prio = Max\{Prio_i, Prio_k \mid T_k$ en attente de SC$\}$. En d'autres termes, si une tâche T_i bloque une tâche T_j, T_i hérite de $Prio_j$ si $Prio_j > Prio_i$. Quand T_i sort de la SC, elle reprend la priorité quelle avait avant d'entrer en SC.

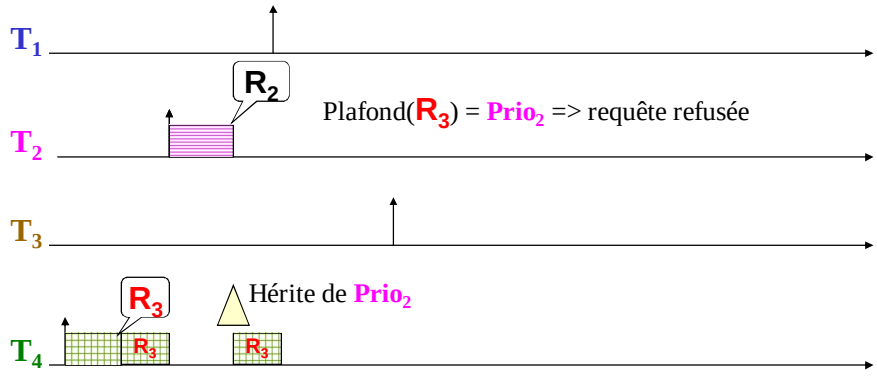
Exemple de fonctionnement de PCP (1/4)



$Prio_4 < Prio_3 < Prio_2 < Prio_1$

Plafond(R_1) = $Prio_1$ Plafond(R_2) = $Prio_2$ Plafond(R_3) = $Prio_2$

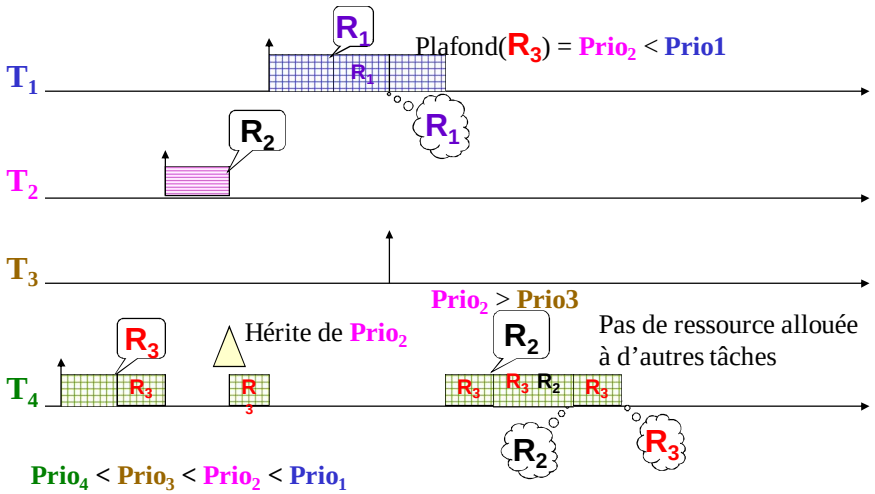
Exemple de fonctionnement de PCP (2/4)



$Prio_4 < Prio_3 < Prio_2 < Prio_1$

Plafond(R_1) = $Prio_1$ Plafond(R_2) = $Prio_2$ Plafond(R_3) = $Prio_2$

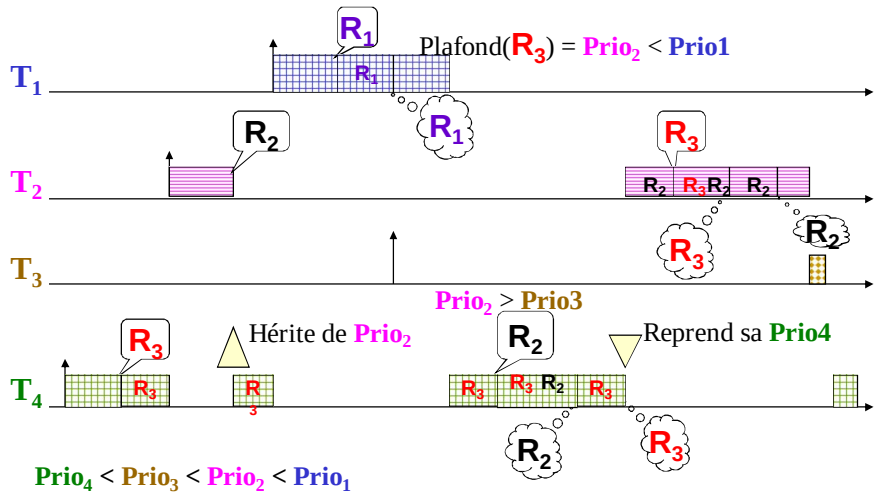
Exemple de fonctionnement de PCP (3/4)



$Prio_4 < Prio_3 < Prio_2 < Prio_1$

Plafond(R_1) = $Prio_1$ Plafond(R_2) = $Prio_2$ Plafond(R_3) = $Prio_2$

Exemple de fonctionnement de PCP (4/4)



$Prio_4 < Prio_3 < Prio_2 < Prio_1$

Plafond(R_1) = $Prio_1$ Plafond(R_2) = $Prio_2$ Plafond(R_3) = $Prio_2$

Propriétés de PCP

- **Utilisable pour les ressources mono exemplaires**
- **Durée de blocage de la tâche la plus prioritaire ≤ pire_durée_SC**

  Avec PCP, la tâche avec la priorité la plus élevée peut être bloquée le temps d'une SC au plus

- **Pour une tâche i :** $B_i = \max_{\forall R, \forall j \in hp(i)} (durée\_SC_j(R))$

  hp(i) : ensemble de tâches plus prioritaires que τ_i. *R* : toute ressource utilisée par une tâche $\in$hp(i).

- **Condition suffisante pour EDF et tâches périodiques** (Chen et Lin 1991)

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

B_i : temps maxi de blocage de la tâche T_i en attente de SC.

Dans la CS, les tâches sont ordonnées selon l'ordre croissant des D_i

- **PCP évite les interblocages**

Protocole à priorité à pile : SRP
(Stack Resource Policy) - (Backer 1991)

- **Principe**

  - Chaque tâche T_i est caractérisée par une priorité $Prio_i$ (statique ou dynamique) et un niveau de préemption π_i (statique).
    π_i se calcule à partir de paramètres tels que D_i, C_i, $Importance_i$ ...
    T_i ne peut préempter T_k que si $\pi_i > \pi_k$

  - Chaque ressource R_x existe en NR_x exemplaires ($NR_x \geq 1$) .
    R_x a un plafond courant C_x égal au niveau de préemption le plus élevé des tâches qui pourraient être bloquées en attente de R_x quand il y a nr_x exemplaires disponibles de R_x :

$$C_x(nr_x) = Max[\ \{0\} \cup \{\pi_k \mid nr_x < \mu_x(T_k)\}\ ]$$

nr_x : Nombre d'exemplaires de R_x actuellement disponibles

$\mu_x(T_k)$: Nombre maxi d'exemplaires de R_x que peut demander la tâche T_k

- Le plafond (Π_s) d'un système avec m ressources partagées est égal au plafond courant le plus élevé des ressources :

$$\Pi_s = Max[C_x \quad x=1, \ldots, m\ ]$$

- Π_s est initialisé à 0 et modifié à chaque acquisition/libération de ressource.
- **Test de préemption** : Si la condition « T_i est la plus prioritaire des tâches prêtes et $\pi_i > \Pi_s$ » est vraie T_i est exécutée, sinon elle reste dans la file des tâches prêtes. Cette condition est testée à chaque fois que Π_s décroît (c-à-d, quand une ressource est libérée).

- **Conséquence du principe précédent**

  - Avec le protocole SRP, une tâche peut être bloquée dès son démarrage et non au moment où elle demande une ressource.
  - Une tâche peut être bloquée par le test de préemption même si elle n'utilise aucune ressource, ce qui permet d'éviter l'inversion de priorité.
  - Le passage du test de préemption par une tâche T_i signifie que les ressources actuellement disponibles sont suffisantes pour satisfaire les besoins de T_i et de toutes les tâches qui pourraient la bloquer. Cela signifie aussi que toute tâche démarrée ne sera plus bloquée à cause d'une ressource. Ainsi, les situations d'interblocage sont évitées.

Exemple SRP (1/5)

3 exemplaires de R_1 1 exemplaires de R_2 3 exemplaires de R_3

T_1 :
demande(1, R_3)
…
demande(1, R_1)
…
libère (1, R_1)
…
libère(1, R_3)

T_2 :
demande(3, R_3)
…
demande(1, R_2)
…
libère (1, R_2)
…
libère (3, R_3)
…
demande(2, R_1)
…
libère(2, R_1)

T_3 :
demande(1, R_2)
…
demande(3, R_1)
…
libère (3, R_1)
…
libère(1, R_2)
…
demande(1, R_3)
…
libère(1, R_3)

$r_1=2, D_1=9, \pi_1=3$ $r_2=1, D_2=16, \pi_2=2$ $r_3=0, D_3=20, \pi_3=1$

Exemple SRP (2/5)

Paramètres de tâches et demandes en ressources

	D_i	π_i	μ_1	μ_2	μ_3
T_1	9	3	1	0	1
T_2	16	2	2	1	3
T_3	20	1	3	1	1

Plafonds dynamiques des ressources
(en fonction du nombre d'exemplaires disponibles)
- : cas impossible

	$C_R(3)$	$C_R(2)$	$C_R(1)$	$C_R(0)$
R_1	0	1	2	3
R_2	-	-	0	2
R_3	0	2	2	3

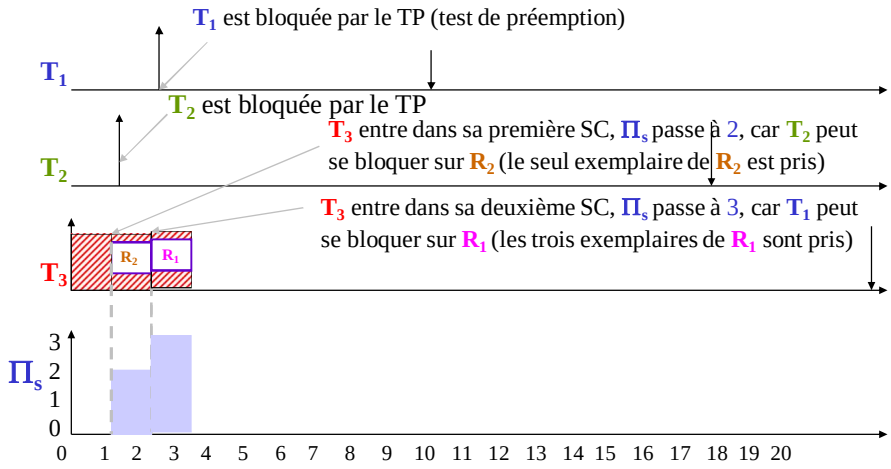
C'est un tableau statique utilisé par le Test de préemption

Lorsque 2 exemplaires de R_1 sont libres, la seule tâche qui pourrait être bloquée est T_3, car elle exige 3 exemplaires de R_1. Par conséquent $C_1(2) = Max[0, \pi_3] = 1$.

Lorsque 1 exemplaire de R_1 est libre, les tâches qui pourraient être bloquées sont T_2 et T_3, car elles exigent chacune plus d'un exemplaire de R_1. Par conséquent $C_1(1) = Max[0, \pi_2, \pi_3] = 2$.

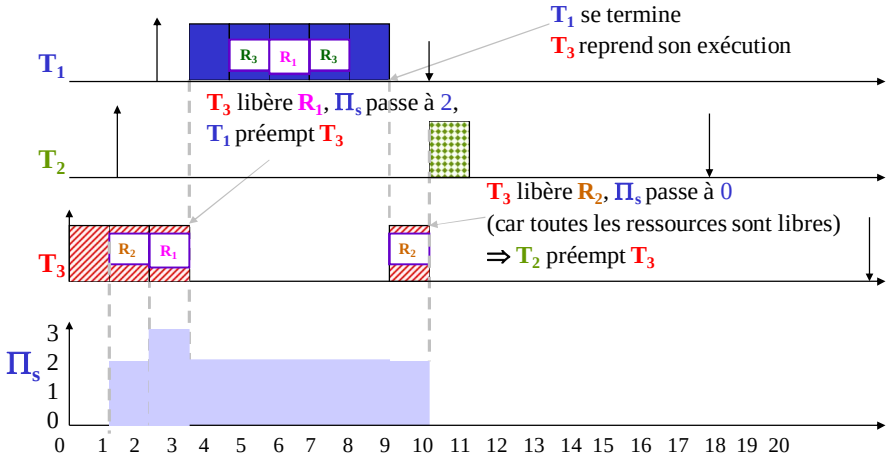
Lorsque aucun exemplaire de R_1 n'est libre, toutes les trois tâches pourraient être bloquées. Par conséquent $C_1(0) = Max[0, \pi_1, \pi_2, \pi_3] = 3$.

Exemple SRP + EDF (3/5)



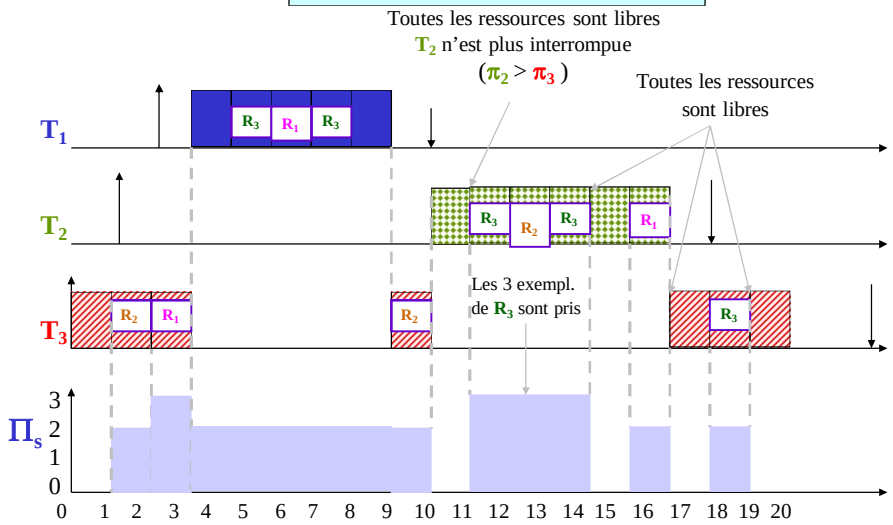
$D_1 = 9, \pi_1 = 3$ **$D_2 = 16, \pi_2 = 2$** **$D_3 = 20, \pi_3 = 1$**

Exemple SRP + EDF (4/5)



$D_1 = 9, \pi_1 = 3$ **$D_2 = 16, \pi_2 = 2$** **$D_3 = 20, \pi_3 = 1$**

Exemple SRP + EDF (5/5)



$D_1 = 9, \pi_1 = 3$ $D_2 = 16, \pi_2 = 2$ $D_3 = 20, \pi_3 = 1$

Propriétés de SRP

- **Utilisable pour les ressources multi exemplaires**
- **Utilisable avec EDF**
- **Durée de blocage ≤ pire_durée_SC**
- **Condition suffisante pour EDF et tâches périodiques** (Baker 1991)

$$\forall k = 1,2,\ldots,n \Rightarrow \left(\sum_{i=1}^{i=k} \frac{C_i}{D_i} \right) + \frac{B_k}{D_k} \leq 1$$

B_i : **temps maxi de blocage de la tâche T_i en attente de SC.**
Dans la CS, les tâches sont ordonnées selon l'ordre croissant des D_i

- **Gère les interblocages**
- **Minimise les changements de contexte**

Comparaison des principaux protocoles de gestion de ressources

	Déclaration préalable des ressources	Prévention d'interblocage	Nombre de blocages	Calcul du de temps de blocage	Moment de blocage
Protocole à héritage de priorité (Priority Inheritance Protocol)	Non (transparent)	Non	Min(n,m)	Difficile	Au Moment de l'accès
Protocole à priorité plafond (Priority Ceiling Protocol)	Oui	Oui	1	Facile	Au Moment de l'accès
Protocole à pile de priorité (Stack Resource Policy)	Oui	Oui	1	Facile	Au moment de la préemption

n : nombre de tâches **m : nombre de ressources**

Quelques résultats [stankovic 95]

- En présence de contraintes d'exclusion mutuelle, il est impossible de trouver un ordonnanceur en ligne optimal.

- Le problème de décider s'il est possible d'ordonnancer des tâches périodiques utilisant les sémaphores pour gérer l'exclusion mutuelle est NP-difficile.

- Le problème d'ordonnancement, avec 2 processeurs, de tâches indépendantes, sans ressources, avec des temps d'exécution quelconques est NP-complet.

- EDF n'est pas optimal dans le cas multiprocesseur.

- Dans le cas de 2 processeurs ou plus, aucun ordonnancement fondé sur les délais (comme EDF par exemple) ne peut être optimal sans connaissances a priori des délais, des temps d'exécution et des instants de réveil.

La plupart des problèmes d'ordonnancement $\in$ NP

$\Rightarrow$ Heuristiques

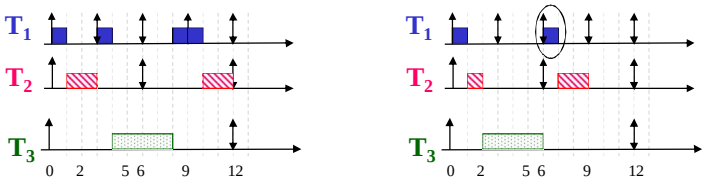
Anomalies de l'ordonnancement

- Ne pas se fier à ce qui semble 'naturel'/'évident'.
- De nombreux types d'anomalies sont publiés.

- Exemple : cas de l'ordonnancement non préemptif à priorité fixe

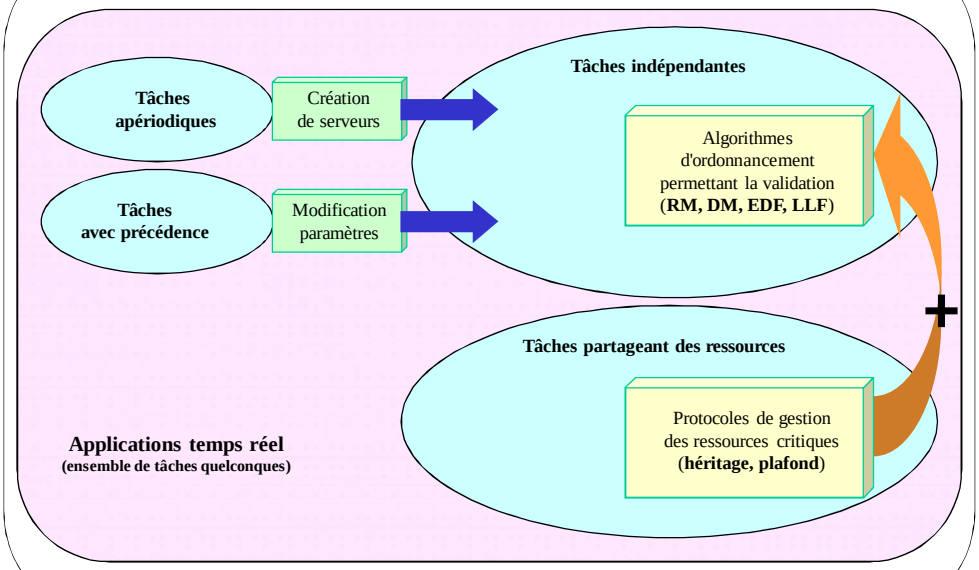
	C_i	P_i
T_1	1	3
T_2	2	6
T_3	4	12

	C_i	P_i
T_1	1	3
T_2	1	6
T_3	4	12



$Prio_1 > Prio_2 > Prio_3$

Conclusion sur l'ordonnancement de tâches (1)



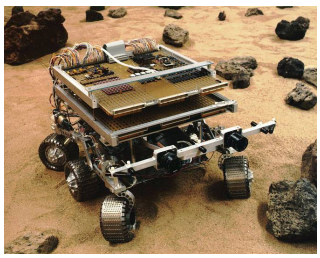
Conclusion sur l'ordonnancement de tâches (2)

- **Détermination efficace des paramètres de tâches (Cmax, Périodes, …)**
- **Recherche de conditions d'ordonnançabilité**
  (difficultés pour les apériodiques, le partage de ressources, la répartition)
- **Meilleure expérimentation des algorithmes**
  (connaissance plus précise de leurs performances)
- **Choix efficace d'un algorithme en fonction des contraintes de tâches**
- **Ordonnancement temps réel et multimédia**
- **Ordonnancement temps réel et réseaux, Internet**
- **Application des techniques approchées (algo génétiques, réseaux de neurones, logique floue …)**

Chapitre 5

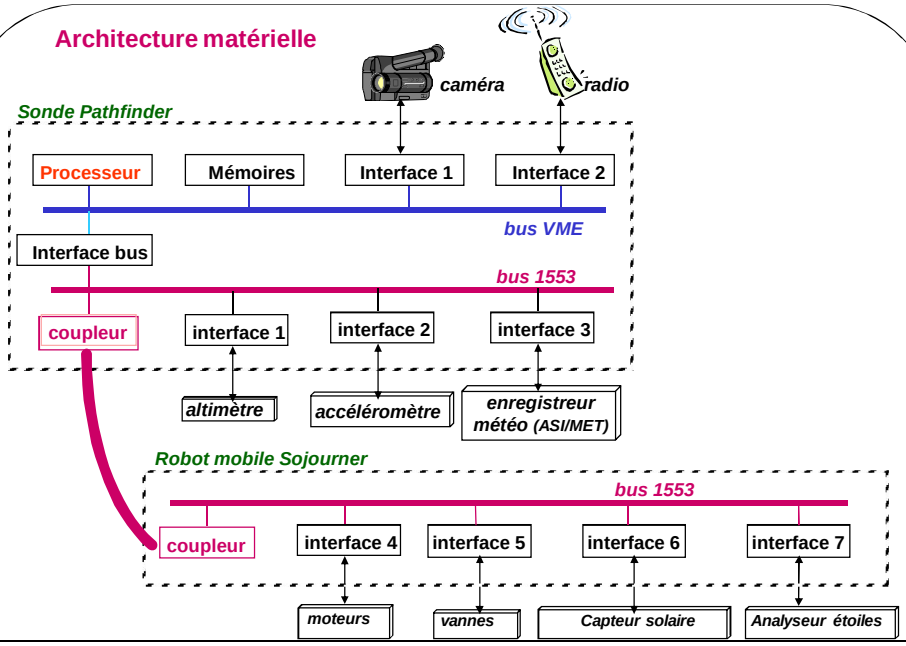
Exemple d'application

Exemple d'application
Exploration de Mars
Mission Pathfinder de la NASA



- Envoi sur Mars d'une sonde d'exploration de 280 Kg.
- Sonde posée sur mars le 4 juillet 1997.
- Libération d'un robot mobile Sojourner (11,5 Kg) se déplaçant à la vitesse de 24 m/h et chargé de faire des mesures (photos, météo, etc.)
- Le robot mobile a commencé son travail le 6 juillet 1997.
- Après quelques jours de fonctionnement correct, le calculateur embarqué s'est réinitialisé à plusieurs reprises conduisant à la perte de nombreuses informations.
- Après une analyse détaillée du problème et une série de tests sur terre, un phénomène d'inversion de priorité a été identifié comme cause de l'erreur. Le code résidant sur la sonde a été modifié (en intégrant un protocole d'héritage de priorité) et a été téléchargé.

Architecture matérielle



Architecture logicielle

- Architecture multitâche gérée par le noyau temps réel VxWorks
- L'application comprend plus de 25 tâches associées à plusieurs étapes de la mission :
  - vol interplanétaire,
  - pose de la sonde sur Mars
  - exploration par le robot mobile
- Communication entre tâches par boîtes aux lettres ou files de messages

Tâches associées à l'exploration par le robot Sojourner

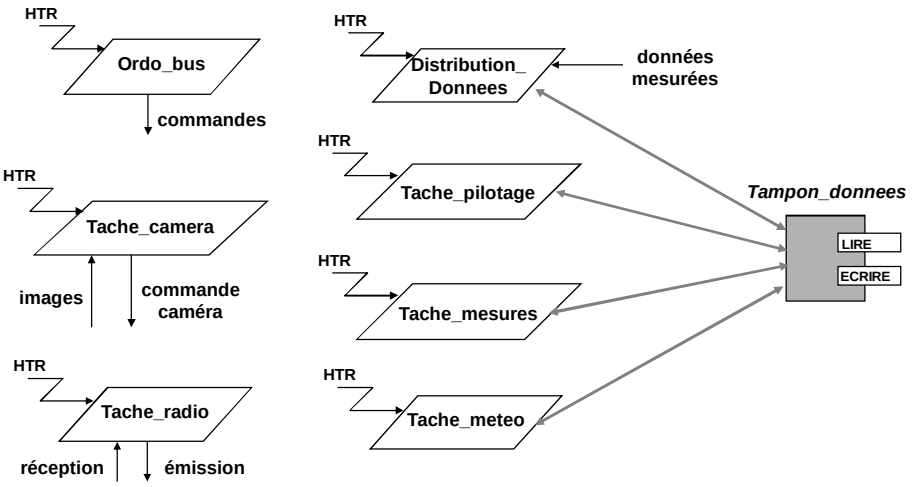
(les paramètres réels de tâches ont été un peu modifiés ici pour les besoins de l'exemple)

Priorité	Nom de tâche	Description	C_i (ms)	P_i (ms)
8	ORDO_BUS	tâche ordonnanceur du bus 1553	25	125
6	DISTRIBUTION _DONNEES	tâche distribution des données du bus 1553	25	125
5	TACHE_PILOTAGE	tâche de pilotage de l'application (robot)	25	250
4	TACHE_RADIO	tâche de gestion des communications radio	25	250
3	TACHE_CAMERA	tâche de gestion de la caméra	25	250
2	TACHE_MESURES	tâche dédiée aux mesures	50	5000
1	TACHE_METEO	tâche de gestion des données météo	{50, 75}	5000

Durée d'exécution variable

$U = 72\%$ si $C_{Tache\_Meteo} = 50$
$U = 72{,}5\%$ si $C_{Tache\_Meteo} = 75$
$\}$ $< 7(2^{1/7} - 1) \Rightarrow$ Ordonnançable avec RM

Toutes les tâches sont périodiques et activées par l'horloge temps réel interne (HTR). Elles partagent une ressource critique commune, appelée TAMPON_DONNEES



Gestion du bus 1553

- **Gestion périodique et pilotée par un signal de fréquence 8 Hz (125 ms de période).**
- **Deux tâches vont permettre de réguler le transfert des données sur ce bus :**
  - **Tâche ORDO_BUS**
    cette tâche, qui possède la priorité la plus haute, vérifie que le transfert de données a été correctement effectué et prépare le nouveau cycle de transfert. Si le transfert ne respecte pas les échéances, une alarme est générée (conduisant à une réinitialisation du système).
  - **Tâche DISTRIBUTION_DONNEES**
    cette tâche, qui possède la troisième priorité la plus haute collecte l'ensemble des dernières données fournies par les différents éléments connectés sur le bus et place ensuite ces données dans la mémoire tampon TAMPON_DONNEES accessible aux autres tâches de l'application.

Hypothèses d'exécution des tâches

- **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 (c'est-à-dire surcoût de demande de ressource négligeable)**
- **Noyau temps réel VxWorks (Wind River Systems)**

Diagramme d'exécution des tâches pour $C_{Tache\_Meteo} = 2$

Séquence valide

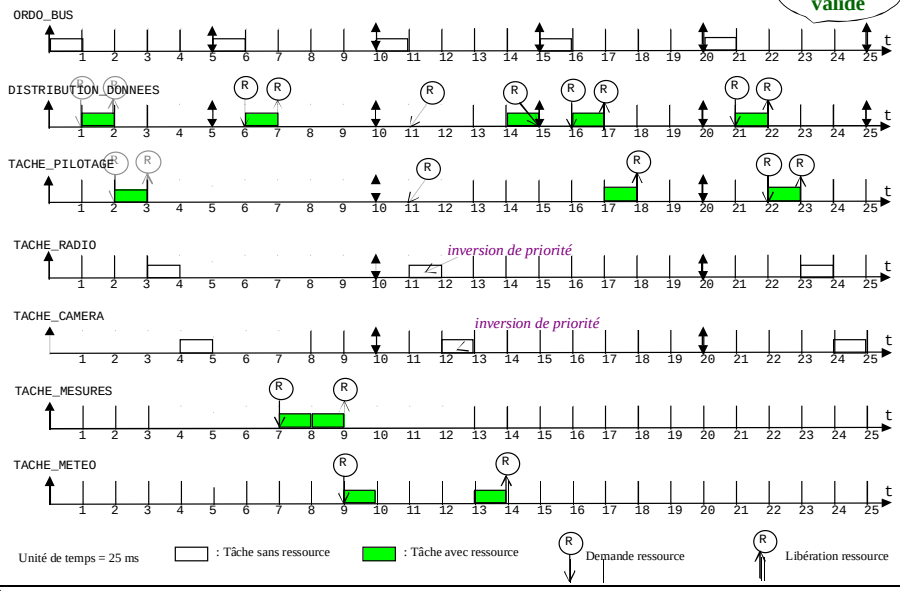


Diagramme d'exécution des tâches pour C_{Tache_Meteo} = 3

Séquence invalide

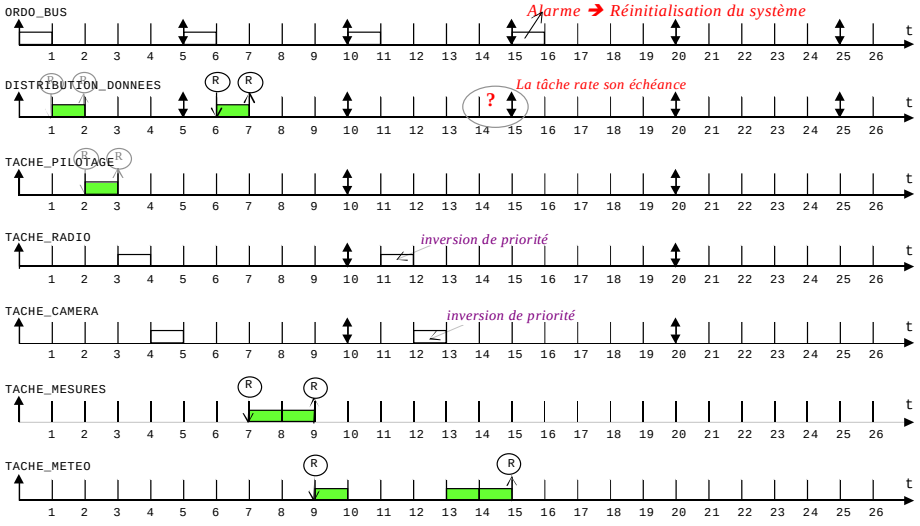
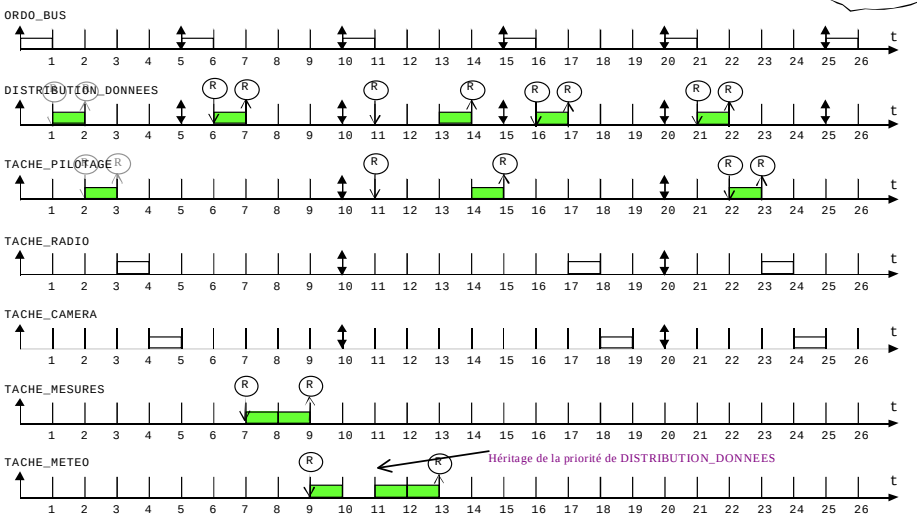


Diagramme d'exécution des tâches pour C_{Tache_Meteo} = 3 avec Protocole d'héritage de priorité

Séquence valide



Explication

Pathfinder contained an "information bus", which you can think of as a shared memory area used for passing information between different components of the spacecraft. A bus management task ran frequently with high priority to move certain kinds of data in and out of the information bus. Access to the bus was synchronized with mutual exclusion locks (mutexes).

The meteorological data gathering task ran as an infrequent, low priority thread, and used the information bus to publish its data. When publishing its data, it would acquire a mutex, do writes to the bus, and release the mutex. If an interrupt caused the information bus thread to be scheduled while this mutex was held, and if the information bus thread then attempted to acquire this same mutex in order to retrieve published data, this would cause it to block on the mutex, waiting until the meteorological thread released the mutex before it could continue. The spacecraft also contained a communications task that ran with medium priority.

Most of the time this combination worked fine. However, very infrequently it was possible for an interrupt to occur that caused the (medium priority) communications task to be scheduled during the short interval while the (high priority) information bus thread was blocked waiting for the (low priority) meteorological data thread. In this case, the long-running communications task, having higher priority than the meteorological task, would prevent it from running, consequently preventing the blocked information bus task from running. After some time had passed, a watchdog timer would go off, notice that the data bus task had not been executed for some time, conclude that something had gone drastically wrong, and initiate a total system reset.
This scenario is a classic case of priority inversion

Chapitre 6

Calcul du pire temps d'exécution (WCET)

Introduction

WCET : Worst Case Execution Time

= Majorant du temps d'exécution

≠ **temps de réponse**

Tâche seule

Toutes les tâches

Pour chaque τ_i
- **Instant d'arrivée**
- **Période**
- **WCET**
- **…**

Ensemble de ressources

Algorithme d'ordonnancement

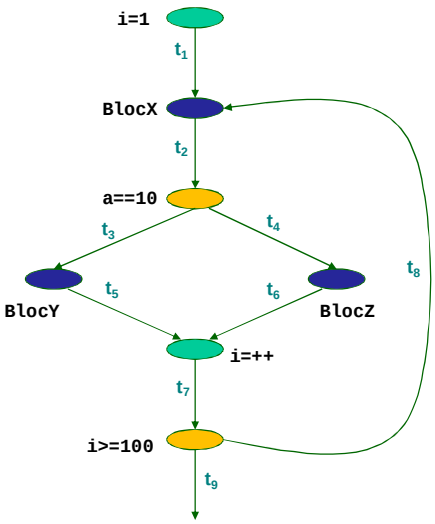
Analyse d'ordonnançabilité

Verdict : O/N

- **Défis pour l'obtention des WCET**
  - Les estimations de WCET doivent être sûres (WCET > tout temps d'exécution possible)
    - Confiance dans les tests de faisabilité
  - Les estimations de WCET doivent être précises
    - Surestimation ⇒ échec potentiel des tests de faisabilité, surdimensionnement des ressources matérielles nécessaires

Facteurs d'influence

```
Void f(int a)
{for (i=1;i<100;i++)
{BlocX;
 if (a==10) BlocY; else BlocZ;
}
```



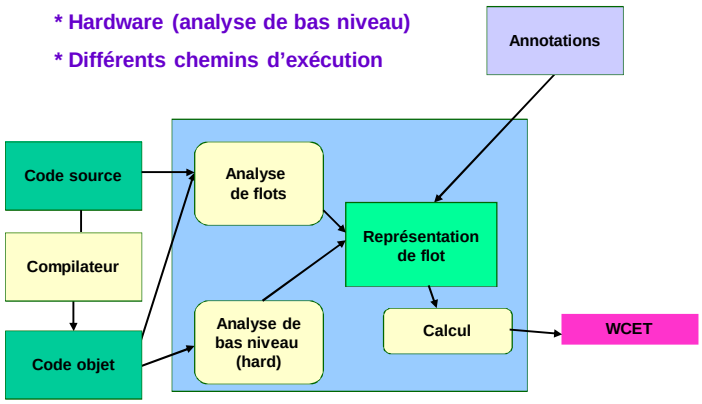
- Chemins d'exécution dépendent de :
  - Structure du programme
  - Valeurs d'entrée
- Durée d'exécution des instructions dépend de :
  - Vitesse du processeur
  - Parallélisme, cache…

Méthodes de calcul du WCET

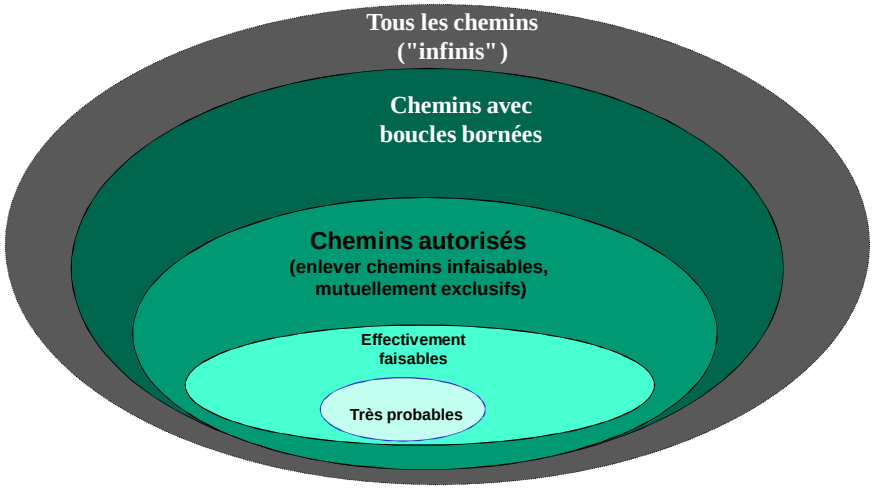
- Méthodes : dynamiques ou statiques
- Méthodes dynamiques
  - ♦ Détermination des jeux d'essai
    - Manuelle, générateurs aléatoires, techniques avancées
  - ♦ Exécution réelle sur la cible
  - ♦ Mesure du pire cas
  - ♦ Risques
    - Pas sûr de trouver le WCET à cause de la non-couverture des jeux d'essai
    - Explosion du nombre de jeux de test
  - ♦ Méthodes les plus répandues dans la pratique (crédibles)

Méthodes statiques

- **Principe :** Analyse du programme sans exécution
- Aspects à analyser
  - * Flots de données
  - * Hardware (analyse de bas niveau)
  - * Différents chemins d'exécution



Analyse de flot (1/3)



Analyse de flot (2/3)

- Chemins infaisables

```
Int Fonc1 (in x) {
  if (X<5)              // A
          X=X+1;        // B
  else X=X*2;           // C
  if (X>10)             // D
    X=X/2;              // E
  return X;             // F
 }
```

Chemin ABCDEF infaisable

- Identification de chemins infaisables ➔ réduction du WCET

Analyse de flot (3/3)

- Nombre maximum d'itérations

```
for (i:=1; i<=N;i++) {
  for (j:=1; j<=i; j++) {
      if  (X<5) X=X+1;
      else X=X+2;
      if (Y>100) Y:= Y/4 ;
      else Y:=Y/2 ;
 }}
```

Complexité des algorithmes

(N+1)N/2 tests
(N+1)N/2 additions
(N+1)N/2 divisions

Typage de données

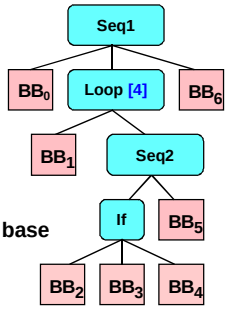
Int N ➔ WCET élevé

Const N = 10 ➔ WCET optimisé

- Nombre maxi d'itérations connu ➔ réduction significative du WCET

Méthode à base d'arbre pour le calcul du WCET (1/2)

- **Structures de données utilisées**
  - Arbre syntaxique du programme
  - Blocs de base

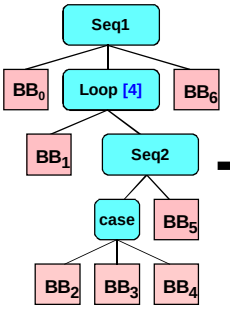


- **Principe**
  - **Détermination du temps d'exécution de chaque bloc de base**
  - **Calcul du WCET par analyse de bas en haut de l'arbre**

Méthode à base d'arbre pour le calcul du WCET (2/2)

Règles de calcul

WCET(SEQ)	S1;…;Sn	WCET(S1) + … + WCET(Sn)
WCET(IF)	if(test) then else	WCET(test) + max(WCET(then) , WCET(else))
WCET(FOR)	for(init;tst;inc) {body}	WCET(init) + maxiter * (WCET(tst)+WCET(body) + + WCET(inc)) + WCET(tst)
. . .		



WCET(Seq1) = WCET(BB0) + WCET(Loop) +WCET(BB6)
WCET(Loop) = 4 * (WCET(BB1) + WCET(Seq2)))
WCET(Seq2) = WCET(If) + WCET(BB5)
WCET(case) = WCET(test) + max(WCET(BB2), WCET(BB3), WCET(BB4))

Difficultés logicielles de calcul du WCET

1. Difficultés liées aux langages

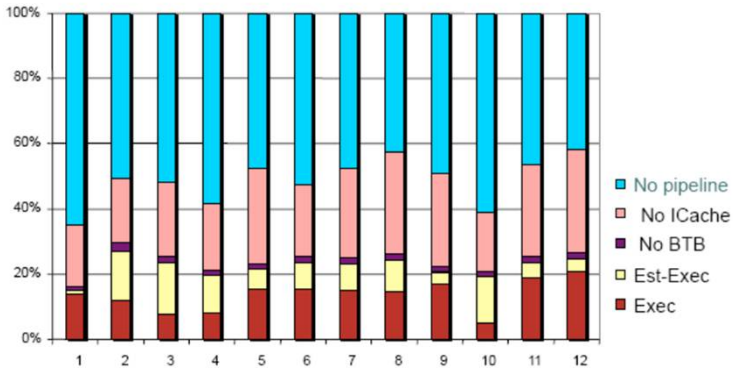
- **Prise en compte de la récursivité**
- **Notion d'objet, appel de méthode**

2. Difficultés liées aux librairies et à l'OS

- **Utilisation de librairies à « la Java » non maîtrisée par le programmeur**
- **Nouvelles architectures logicielles (composants, produits sur étagère)**
- **Allocation dynamique de mémoire, ramasse-miette**
- **Gestion de mémoire virtuelle**
- **Autres**

Considérations matérielles dans le calcul du WCET

- **Processeurs simples**
  - Temps d'exécution d'une instruction dépend uniquement de son type et de ses opérandes
  - Pas de recouvrement entre instructions, pas de hiérarchie de mémoire
- **Processeurs complexes**
  - Effets locaux
    - Recouvrement entre instructions (notion de pipeline)
  - Effets globaux
    - Caches de données,
    - Caches d'instructions,
    - Prédicteurs de branchement
    - …



Exemple de surdimensionnement du WCET

(projet Heptane-IRISA)

- **BTB (branch transfer buffer)**
- **Exec : exécution réelle**
- **Est-exec : estimation d'exécution**

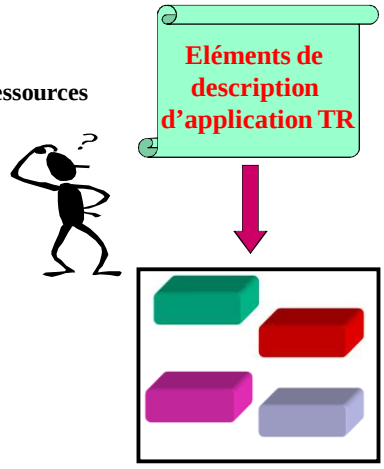
Chapitre 7

Validation d'applications temps réel

1. Eléments de description d'applications TR

➔ **Une application TR est définie par :**

- **Charge demandée par les tâches**
- **Relations de précédence et partage de ressources**
- **Autres caractéristiques des tâches**



➔ **Une architecture support :**

- **Algorithmes d'ordonnancement**
- **Protocoles de gestion des ressources**
- **S/système de communication TR**
- **Noyau de système**
- **Processeur(s)**
- **S/système d'E/S**
- **Autres s/systèmes**

2. Validation

➔ **Objectif**

Démontrer/montrer par une procédure de preuve, de simulation ou de test que les choses sont « bonnes/correctes/cohérentes ».

➔ **3 étapes essentielles pour y parvenir**

1. Vérifier que les CT sont compatibles (cohérentes) avec le cahier des charges et que les CT des différents composants (tâches/messages) sont cohérentes entre elles

2. Vérifier que tout composant peut respecter ses CT s'il exécute seul et qu'il a toutes les ressources requises (ordonnançabilité individuelle)

3. Vérifier que toute l'application respecte ses CT en tenant compte de l'architecture matérielle et logicielle support (ordonnançabilité collective).

Analyse d'ordonnançabilité

- L'analyse d'ordonnançabilité = étape primordiale pour les applications TR critiques
- Objectif : Eliminer les risques de fautes temporelles
- Calcul de divers temps
  - temps d'exécution
  - temps de changement de contexte
  - temps de réponse
  - temps de communication
  - temps de blocage
- Utilisation des conditions nécessaires et/ou suffisantes (preuve)

  Connaissance des propriétés (optimalité, conditions d'ordonnançabilité, …) des algorithmes requise
- Simulation
- Coût : Le problème de l'ordonnancement est NP-difficile

Paramètres d'analyse d'ordonnançabilité

- Etude des phénomènes apériodiques
  - Difficulté de prédire le respect des CT dans le cas général
  - Bonne maîtrise des lois de distribution des instants d'arrivée des événements apériodiques qui déclenchent les tâches/messages
  - Utiliser la simulation pour mieux maîtriser/appréhender les phénomènes aléatoires
- Etude des phénomènes périodiques
  - Informations connues de manière statique ⇒ Validation plus facile
  - Analyser l'exécution des tâches pendant un intervalle de temps représentatif

  Période d'étude = Cycle majeur = Méta-période = H

  H : SI $\forall i,j\ r_i = r_j$ (départ simultané des tâches) = PPCM$\{P_i, i=1, \ldots, n\}$

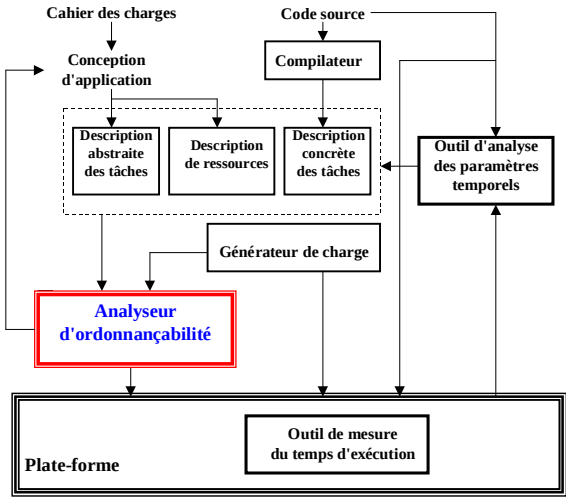
  SINON = max$\{r_i\}$ + 2×PPCM$\{P_i, i=1, \ldots, n\}$

3. Outils pour l'analyse d'ordonnançabilité

- **PERTS (Prototyping Environment for Real-Time Systems)**
  University of Illinois
  commercialisé par Tri-pacific software
  Rapid RMA et Rapid SIM
  http://www.tripac.com/

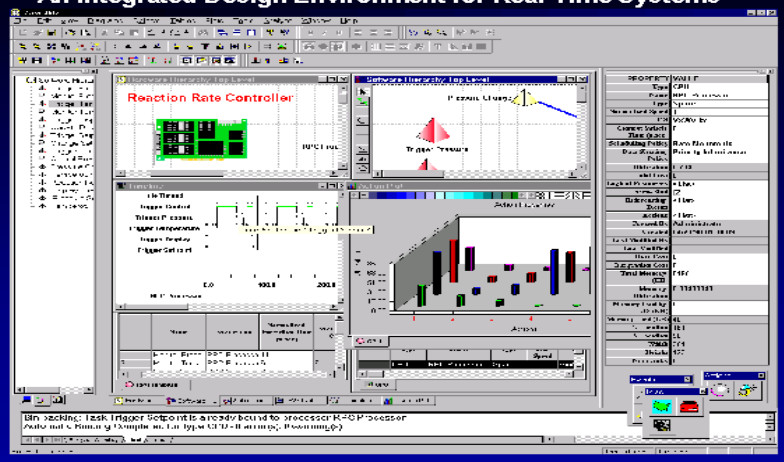
- **TimeWiz de TimeSys Corp.**
  http://www.timesys.com/

PERTS



TimeWiz™

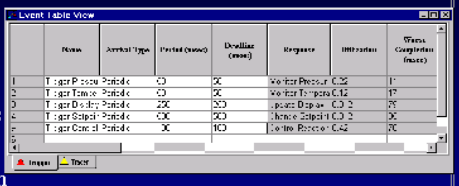
An Integrated Design Environment for Real-Time Systems



TimeWiz is a unique and powerful product that lets you represent hardware and software, analyze and simulate the timing behavior of your system, model processors and networks for end-to-end performance, chart your system parameters and generate integrated system reports. All nicely integrated and affordable.

Action Table View

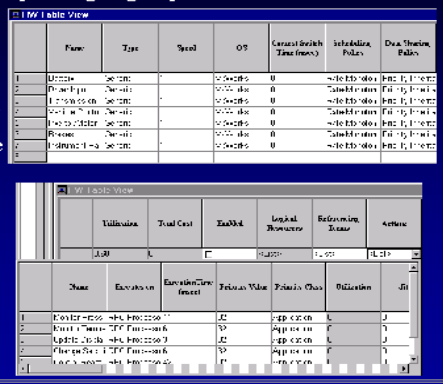
- The *Action Table View* can be used to view/enter/modify values for trigger and tracer events
- Examples:
  - Arrival type
  - Period
  - Deadline
  - Response actions
  - Utilization
  - Worst completion



TimeSys Corporation Real-Time... Real Solutions

Hardware Table View

- The *Hardware Table View* is used to view/enter/modify values for different properties corresponding to groups of TimeWiz resource objects
- Examples:
  - Scheduling policy
  - Data-sharing policy
  - Context switch time
  - Logical resources
  - Utilization

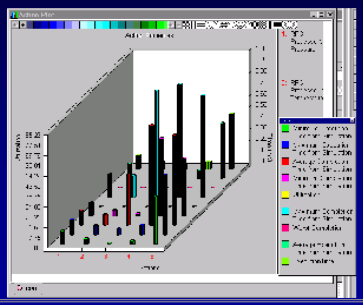


TimeSys Corporation

Real-Time... Real Solutions

Viewing Simulation Results

- Results of the simulation can also be viewed using the plot view



TimeSys Corporation

Real-Time... Real Solutions

4. Conclusion

- **Variété d'algorithmes d'ordonnancement ⇒**
  Des profils d'ordo selon les classes de problèmes
- **Développer des outils d'aide au choix des algorithmes selon le profil considéré**
- **Développer des outils d'assistance à la validation d'applications TR.**

Ouvrages

- **Burns A., Wellings A.,** ***Real-time systems and programming languages.*** **Addison-Wesley 1997.**
- **Buttazzo G.C.,** ***Hard real-time computing systems, predictable scheduling,***
  algorithms and applications, Kluwer Academic Publishers, 2004.
- **Cottet F., J. Delacroix J., Kaiser C. et Mammeri Z.,** ***Ordonnancement temps réel*****, Hermès 2000.**
- **Kopetz K.,** ***Real-Time Systems. Design Principles for Distributed Embedded Applications.***
  Kluwer Academic Publishers 1997
- **Liu J.W.S., Real-time systems, Prentice Hall 2000.**
- **Rajkumar R.,** ***Synchronization in Real-Time Systems. A priority Inheritance Protocol.***
  Kluwer Academic Publishers 1991
- **Stankovic J.A., Spuri M., Ramamritham K., Buttazzo G.** ***Deadline Scheduling for Real-Time Systems. EDF and related Algorithms.*** **Kluwer Academic Publishers 1998**

Articles

- **Agrawal G. et al., "Local synchronous capacity allocation schemes for guaranteeing messages deadlines with the timed token protocol",** ***Proceedings of INFOCOM'93, San Francisco*** **(1993), p. 186-193.**
- **Aras C., Kurose J.F., Reeves D.S., and Schulzrinne H., "Real-time communication in packet switched networks",** ***Proceedings of the IEEE*****, vol. 82, n° 1, p. 122- 139, 1994.**
- **Cardeira C. et Mammeri Z., "Ordonnancement de tâches dans les systèmes temps réel répartis" ,** ***RAIRO-APII*****, vol. 28, 4, p. 353-384, 1994.**
- **A. Colin et al., Calcul de majorants de pires temps d'exécution: état de l'art. Revue Technique et Science Informatiques, Numéro spécial Temps-réel, 2003.**
- **Cottet F., J. Delacroix J., Kaiser C. et Mammeri Z., "Ordonnancement temps réel – Ordonnancement centralisé",** ***Collection Techniques de l'ingénieur, Traité Mesures et contrôle,*** **Article** ***R8055*****, 1999.**
- **Cottet F., J. Delacroix J., Kaiser C. et Mammeri Z., "Ordonnancement temps réel – Ordonnancement réparti",** ***Collection Techniques de l'ingénieur, Traité Mesures et contrôle,*** **Article** ***R8056*****, 2000.**
- **Malcolm N., and Zhao W., "Hard real-time communication in multiple-access networks",** ***Journal of Real-Time Systems*** **(8): 35-77, 1995.**
- **Y. Manabe, S. Aoyagi, " A feasibility decision algorithm for rate monotonic and deadline monotonic scheduling", Journal of Real-Time Systems, 14(2):171-181, 1998**
- **I. Ripoli, A. Crespo, A. Mok, "Improvement in fesability testing for real-time tasks", Journal of Real-Time Systems, 11(1):19-39, 1996.**
- **Sha L., Rajkumar R., and Lehoczky J.P., "Priority inheritance protocols: An approach to real-time synchronisation",** ***IEEE Transactions on Computers*****, vol. 39, n° 9, 1990, p. 1175-1185.**
- **Sprunt B., Sha L., and Lehoczky J.P., "Aperiodic task scheduling for hard-real-time systems",** ***The Journal of Real-Time Systems*****, p. 27-60, 1989.**
- **Stankovic J.A., "Misconceptions about real-time computing",** ***Computer*****, 21, p. 10-19, 1988.**
- **Stankovic J.A., Spuri, M., Di Natale M., and Buttazzo G.C., "Implications of classical scheduling results for real-time systems",** ***IEEE Computer*****, vol. 28, 8, p. 16-25, 1995.**