

Option STR  
Systèmes temps-réel

Isabelle PUAUT  
puaud@irisa.fr  
www.irisa.fr/caps/people/puaud/puaud.html







Définition

- Système embarqué : définition
  - Fait partie d'un dispositif qui n'est pas un ordinateur (système **enfoui**)
  - En général
    - Utilise de manière conjointe du **matériel** et du **logiciel** pour mettre en œuvre une fonction spécifique
      - Le logiciel est utilisé pour plus de **flexibilité**
      - Le matériel (processeur, ASIC, DSP, mémoire) est utilisé pour plus de **performances**
    - Interagit fortement avec son environnement

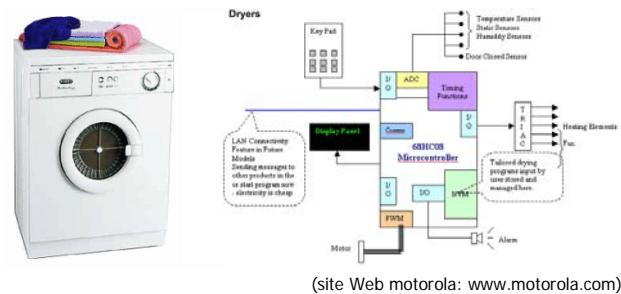
Systèmes temps-réel embarqués  
Caractéristiques générales

- Caractéristiques typiques
  - Fonctions mises en œuvre **spécifiques** au domaine d'application du système (pas généraliste - « general purpose »)
  - Complexité accrue des systèmes embarqués
    - de plus en plus de besoins en **performances** et en **temps-réel**
- Processeurs dédiés peuvent être des composants clés (DSP, décodeurs matériels)

Systèmes temps-réel embarqués  
Domaines d'application

- Produits de grande consommation
  - Cafetière, machines à laver, fours à micro-onde 
- Electronique grand public
  - Caméras numériques, appareils photo numérique 
  - Multimédia, téléphonie : décodeurs vidéo, téléphones portables, PDA, consoles de jeu 
- Automobile
  - Systèmes anti-blocage de freins, contrôle moteur, informatique de confort 
- Contrôle de procédés industriels
- Avionique, spatial, production d'énergie (nucléaire) 
- Périphériques informatique : FAX, imprimantes 
  - Grande **diversité des besoins** (puissance, fiabilité, criticité)

Systèmes temps-réel embarqués  
Exemple : sèche-linge

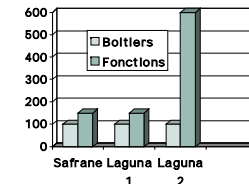


Option STR – 2006-2007

5

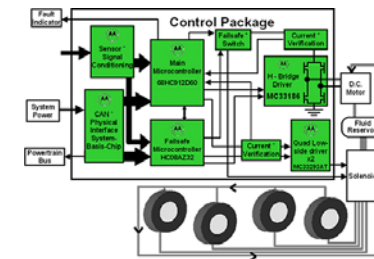
Systèmes temps-réel embarqués  
Exemple : automobile

- Evolutions des fonctions logicielles
- Système antiblocage (ABS)



(C. Balle, Journée Illiatech, octobre 2001)

Option STR – 2006-2007

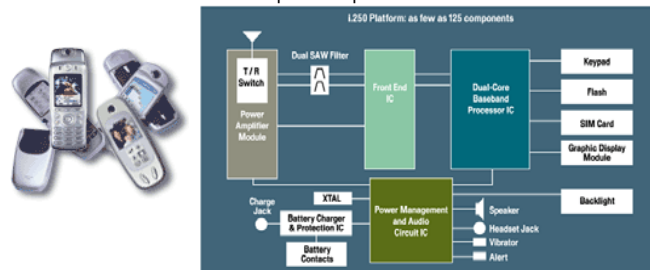


(site Web motorola : [www.motorola.com](http://www.motorola.com))

6

Systèmes temps-réel embarqués  
Exemple : téléphone portable

- Plate-forme I.250 Motorola pour téléphones cellulaires



(site Web motorola : [www.motorola.com](http://www.motorola.com))

Option STR – 2006-2007

7

Systèmes temps-réel embarqués  
Le marché de l'embarqué

- A l'heure actuelle
  - Processeurs embarqués / microcontrôleurs : 31 M\$ par an
  - Processeurs généralistes : 46 M\$ par an
- Evolutions du marché
  - Processeurs embarqués : + 18 % par an
  - Processeurs généralistes : + 10 % par an
- Prévisions : le marché de l'embarqué va dépasser celui des processeurs généralistes dans la décennie
  - Complexité des codes et des architectures croissantes

(R. Gupta, UC Irvine, cours « embedded systems »)

Option STR – 2006-2007

8

## Systèmes temps-réel embarqués Contraintes de conception (1/2)

- Temps-réel
  - Le temps **réel** (physique) entre dans le critère de correction des applications
  - Besoin de **garanties** sur le respect de contraintes temporelles (prévisibilité)
- Sécurité de fonctionnement
  - Définition : capacité à placer une confiance justifiée dans le service
  - Entraves : défaillances provenant de **fautes**
  - Moyens pour la sécurité de fonctionnement (prévention des fautes, **tolérance aux fautes**, prévision des erreurs)
  - Large gamme de méthodes selon le type de fautes et la criticité des applications (matériel, logiciel)
  - Systèmes critiques : **certification**

Option STR – 2006-2007

9

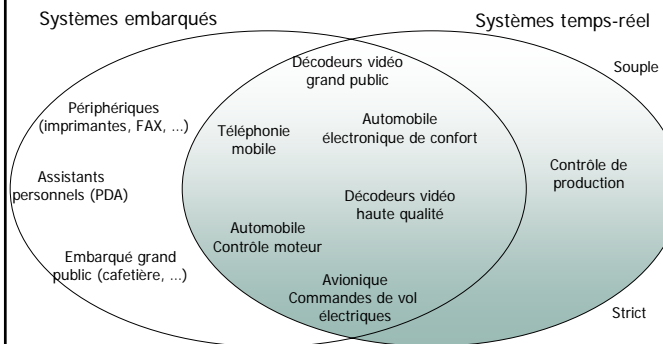
## Systèmes temps-réel embarqués Contraintes de conception (2/2)

- Coût
  - Coût (temps) de conception (time-to-market).
  - Coût d'achat : surface de silicium
    - **Ressources limitées (mémoire)**
    - **Energie limitée (coût des batteries)**
  - Processeurs moins élaborés (plus simples) dans l'embarqué que les processeurs généralistes (coût + prévisibilité)
  - Ergonomie : contrainte de poids
- Lien entre toutes ces contraintes
  - Ex : besoin en performance vs temps-réel et consommation
  - Exploration multi-critères pour les choix matériel - logiciel

Option STR – 2006-2007

10

## Temps-réel vs embarqué



Option STR – 2006-2007

11

## Systèmes temps-réel embarqués Plan de l'option

- Aspects temps-réel (3/4 du cours)
  - [Introduction aux systèmes temps-réel](#)
  - [Ordonnancement temps-réel](#)
    - Classification, quelques politiques d'ordonnancement
  - [Exécutifs temps-réel \(multitâches\)](#)
    - Motivations, rôle et mécanismes de base
    - Ordonnancement
    - Synchronisation, communication, gestion de la mémoire
    - [Quelques exécutifs temps-réel](#)
  - [Analyse d'ordonnancabilité et obtention de pires temps d'exécution](#)
  - Ordonnancement hybride de [tâches temps réel strict et non strict](#)
- [Minimisation de la consommation énergétique](#)
- [Etude de cas](#)

Option STR – 2006-2007

12

## Systèmes temps-réel embarqués Non traités dans cette option

---

- ❑ Tolérance aux fautes
- ❑ Aspects parallélisme / distribution
- ❑ Méthodologies de conception et outils de conception associés
  - (UML-RT, HRT-HOOD)
- ❑ Aspects langage de programmation, programmation synchrone
  - PTR, U.E. libre master 2 professionnel
- ❑ Méthodes de validation des logiciels temps-réel embarqués (méthodes formelles - model checking), test, simulation
- ❑ Traitement du signal, échantillonnage, conception conjointe HW-SW (co-design), matériels dédiés

## Systèmes temps-réel embarqués Quelques références bibliographiques

---

- ❑ A. Burns et A. Wellings. *Real-Time Systems and Programming Languages*, second edition, Addison Wesley
- ❑ G. C. Buttazzo. *Hard Real-Time Computing Systems, predictable scheduling and applications*, Kluwer academic publishers
- ❑ F. Cottet, J. Delacroix, C. Kaiser, Z. Mammeri. *Ordonnancement temps-réel, cours et exercices corrigés*, Hermes
- ❑ J. Liu. *Real-Time Systems*. Prentice Hall
- ❑ H. Kopetz. *Real-Time Systems*. Kluwer academic publishers

---

## Introduction aux systèmes temps-réel

---

## Notion de temps-réel

---

- ❑ Temps-réel
  - Le temps **réel** (physique) entre dans le critère de correction des applications en plus de leur correction fonctionnelle
- ❑ Définition d'une contrainte de temps : limite **quantifiée** sur le temps séparant deux événements (limite min et/ou max)
  - Quantifiée = en rapport avec le **temps réel** (environnement extérieur)
  - Ex : échéance de terminaison au plus tard (deadline) = limite maximale entre arrivée d'un calcul (tâche) et sa terminaison

## Exemples de contraintes de temps

- Échéance de terminaison au plus tôt ou au plus tard (deadline)
  - Ex : synchronisation son-image
- Cohérence entre instants de production des résultats
  - Ex : synchronisation son-image
- Cadence de production des sorties
  - Ex : régularité de présentation des images dans une vidéo

Option STR – 2006-2007

17

## Classes de systèmes temps-réel

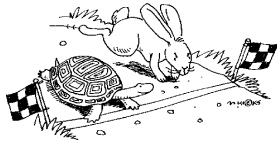
- **Temps-réel strict/dur** (hard real-time) : le non respect d'une contrainte de temps a des conséquences graves (humaines, économiques, écologiques) : besoin de garanties
  - Ex : applications de contrôle dans l'aviation, le spatial, le nucléaire, contrôle de production de matériaux toxiques
  - On se mettra par défaut dans ce cadre
- **Temps-réel souple/mou** (soft real-time) : on peut tolérer le non respect occasionnel d'une contrainte de temps (garanties probabilistes)
  - Ex : applications multimédia grand public (vidéo à la demande, TV)
- Applications interactives : temps de réaction invisible à l'utilisateur
  - Pas temps-réel tel qu'on l'entend dans ce cours

Option STR – 2006-2007

18

## Mauvaises interprétations de la notion de temps-réel

- « Real-time is not real-fast » ou « rien ne sert de courir, il faut partir à point »



- Il existe des contraintes strictes qui ne sont pas courtes : dépend de l'environnement
- La puissance de calcul : condition nécessaire mais pas suffisante
- Vitesse moyenne ou vitesse maximale n'est pas importante, le **pire-cas** importe
  - Murphy's law : if something can go wrong, it will go wrong

Option STR – 2006-2007

19

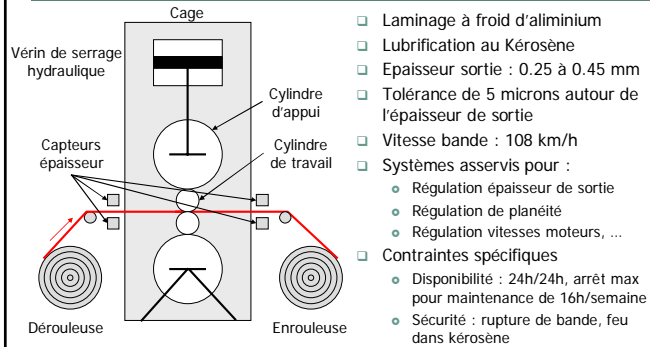
## Points clés pour la réalisation d'applications temps-réel

- Déterminisme d'exécution (prévisibilité, predictability)
  - On doit connaître **tous** les calculs effectués (application, OS) : charge
  - On doit connaître leur temps de calcul de manière sûre (non sous-estimée) -> matériel prévisible
- Choix de l'ordre d'exécution des fonctions important
  - **Ordonnement des calculs**. Pas « à la main » : **méthodes d'ordonnement** connues avec résultats théoriques associés (optimalité)
- Preuve a priori des contraintes de temps à partir de la charge de travail (arrivée des calculs, durée traitement, ...) :
  - Le test n'est pas suffisant si on veut un grand niveau de garantie
    - Ex : Février 91, erreur de prédiction de trajectoire d'un missile Scud en Arabie Saoudite. Dérive d'horloge accumulée sur 100h, les tests n'ayant jamais été aussi loin : victimes civiles
  - **Conditions (analyse) d'ordonnabilité**

Option STR – 2006-2007

20

## Exemple d'application temps-réel (1/2) Le laminoir [Cottet, 2000]



Option STR – 2006-2007

21

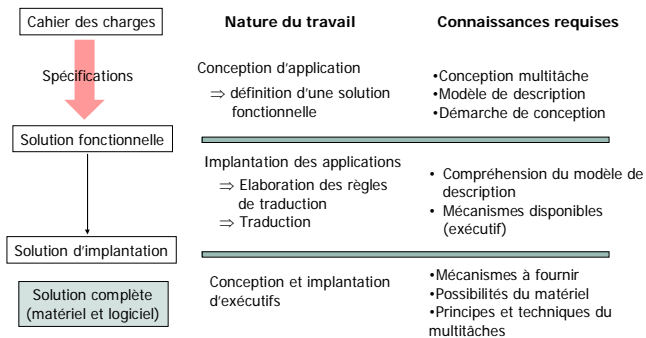
## Exemple d'application temps-réel (2/2) Le laminoir [Cottet, 2000]

- Système d'acquisition et analyse en temps-réel
  - Sous-ensemble du système de contrôle informatique
  - Rôle : traitement et stockage en temps-réel des signaux provenant des calculateurs de pilotage du laminoir
- Les signaux : essentiellement périodiques
  - Compensation des faux-ronds : 984 octets toutes les 4 ms
  - Arrivée d'une nouvelle bobine : 160 octets toutes les 3 mn
  - Régulation de serrage des cylindres du laminoir : 544 octets toutes les 20 ms
  - Planéité en sortie : 2052 octets toutes les 100 ms
- Contraintes de temps proviennent de l'environnement

Option STR – 2006-2007

22

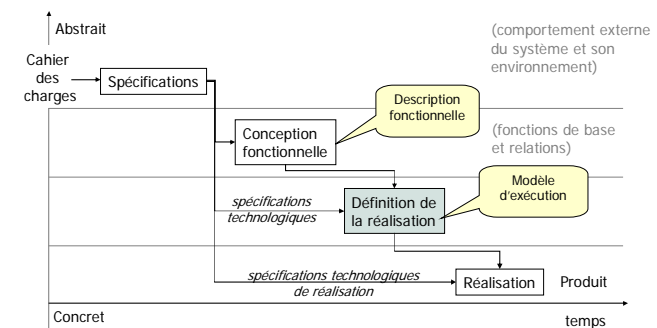
## Niveaux d'approche pour la conception d'applications (temps-réel)



Option STR – 2006-2007

23

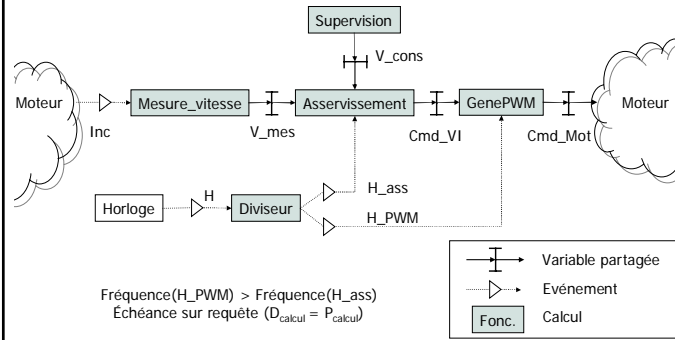
## Situation dans le processus de développement (1/2)



Option STR – 2006-2007

24

## Situation dans le processus de développement (2/2) Exemple de description fonctionnelle



Option STR – 2006-2007

25

## Ordonnement temps-réel

## Introduction (1/2)

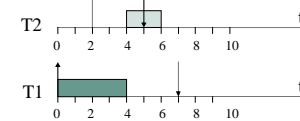
- Définition
  - Ensemble des règles définissant l'ordre d'exécution des calculs sur le processeur
- Pourquoi ordonnancer ?
  - Parce que ça a un impact sur le respect des contraintes de temps
  - Exemple :
    - Tâche T1 : arrivée en 0, temps d'exécution 4, échéance 7
    - Tâche T2 : arrivée en 2, temps d'exécution 2, échéance 5
    - Ordonnancement O1: **premier arrivé, premier servi, non préemptif**
    - Ordonnancement O2: **préemptif à priorité, T2 plus prioritaire que T1**
    - Échéance des tâches respectées avec O2, pas avec O1

Option STR – 2006-2007

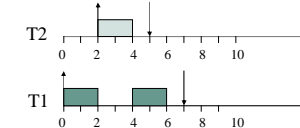
27

## Introduction (2/2)

- Ordonnancement O1 : T2 rate son échéance



- Ordonnancement O2 : toutes les échéances sont respectées



T=2 : préemption de T1  
au profit de T2

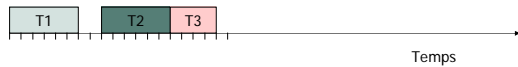
Option STR – 2006-2007

28

Classification (1/5)  
Ordonnancement « hors-ligne » / « en-ligne »

□ Hors-ligne

- La séquence d'ordonnancement est **pré-calculée** avant l'exécution effective (on dit aussi « **time-driven** » scheduling)
- A l'exécution, l'ordonnanceur est un simple **séquenceur** (« cyclic scheduler ») : pas besoin d'exécutif multitâche



□ Evaluation

- Intérêts
  - Mise en œuvre **simple**
  - Facile de détecter les dépassements d'échéances (surcharge)
- Limitations : **rigidité** (pas toujours applicable)

Option STR – 2006-2007

29

Classification (2/5)  
Ordonnancement « hors-ligne » / « en-ligne »

□ Ordonnancements « en-ligne »

- Les décisions d'ordonnancement sont prise **au cours** de l'exécution
- A l'exécution, l'ordonnanceur implante un **algorithme d'ordonnancement** permettant de savoir à tout instant quel tâche exécuter : besoin d'un exécutif multitâche
- Généralement, ordonnancements conduits par la **priorité** (voir plus loin)

□ Evaluation

- Intérêts : **flexibilité**
- Inconvénients :
  - **surcoûts** de mise en œuvre
  - plus difficile de détecter les surcharges

Option STR – 2006-2007

30

Classification (3/5)  
Ordonnancement non préemptif / préemptif

□ Non préemptif

- On n'interrompt **jamais** l'exécution d'une tâche en cours au profit d'une autre tâche

□ Préemptif

- La tâche en cours **peut perdre** involontairement le processeur au profit d'une autre tâche (jugée plus urgente)

□ Préemptif ⇒ besoin d'un exécutif multitâche

□ Remarques : orthogonal par rapport à la classification en-ligne / hors-ligne

- Un ordonnancement hors-ligne peut être préemptif
- Un ordonnancement en-ligne peut être non préemptif

Option STR – 2006-2007

31

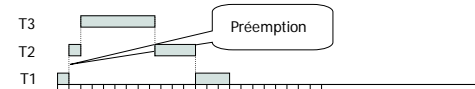
Classification (4/5)

□ Ordonnancement conduits par la **priorité**

- Ordonnancements préemptifs
- C'est la tâche de plus haute priorité qui s'exécute à tout instant
- Priorités **fixes** (statiques) / priorités **dynamiques**
  - Fixes : indépendants du temps, fixées a priori
  - Dynamiques : évoluent avec le temps, au prix d'un surcoût à l'exécution

□ Exemple d'exécution d'un ordonnancement conduit par la priorité

- $prio(T3) > prio(T2) > prio(T1)$  (ici, priorités fixes)
- T1, T2 et T3 arrivent respectivement aux dates 1, 2, et 3



Option STR – 2006-2007

32

## Classification (5/5)

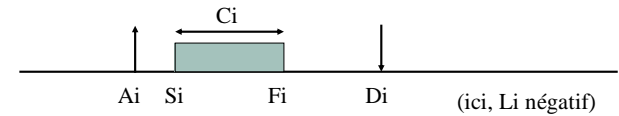
- Optimal vs heuristique
  - Optimal : si ne trouve pas de solution au problème d'ordonnement posé, alors aucun autre algorithme de la même classe ne peut en trouver

Option STR – 2006-2007

33

## Notations (1/2)

- Arrivée  $A_i$
- Temps de calcul maximum sans préemption  $C_i$
- Echéance (deadline) : relative ou absolue  $D_i$
- Date de début (start time)  $S_i$
- Date de fin (finish time)  $F_i$
- Retard (lateness)  $L_i = (F_i - D_i)$
- Laxité (slack time)  $x_i(t) = D_i - (t + C_i - c_i(t))$   
 $X_i = x_i(A_i) = (D_i - A_i - C_i)$



Option STR – 2006-2007

34

## Notations (2/2)

- Arrivées des tâches
  - Périodiques : arrivée à intervalles réguliers ( $P_i$ )
    - Date d'activation initiale, offset  $O_i$
    - Si pour tout  $i, j$   $O_i = O_j$ , tâches synchrones
    - Si  $D_i = P_i$ , tâche à échéance sur requête
  - Sporadiques : on connaît une borne minimale sur l'intervalle entre deux arrivées
  - Apériodiques : tout ce qui ne rentre pas dans les deux catégories précédentes
- Synchronisations (donc blocages potentiels)
  - Ressources partagées
  - Précédences

Option STR – 2006-2007

35

## Quelques ordonnancements

- Ordonnements non préemptifs
  - Selon l'ordre d'arrivée : premier arrivé – premier servi (FIFO)
  - Selon la durée de calcul : plus court d'abord
  - Exploration des ordres possibles [Bratley, 1971, Spring, 1987]
    - Exploration exhaustive (branch-and-bound) : complexité algorithmique (problème NP-complet)
    - Recherche orientées ou heuristiques pour réduire la complexité
- Ordonnements préemptifs
  - Sans notion de priorité : temps partagé avec politique du tourniquet (round-robin)
  - Selon priorité (statiques ou dynamiques) : c'est la tâche la plus prioritaire qui s'exécute à tout instant

Option STR – 2006-2007

36

### Ordonnement préemptif à priorité fixe Rate Monotonic (1/2)

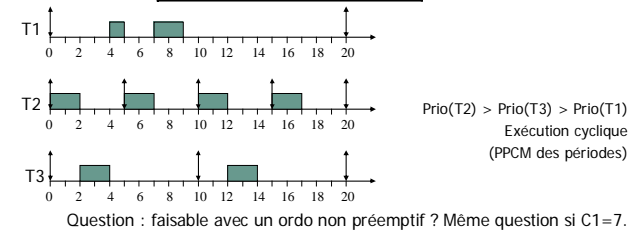
- Pour tâches périodiques
- Définition [Liu & Layland, 1973]
  - La priorité d'une tâche est inversement proportionnelle à sa période d'activation (conflits résolus arbitrairement)
- Propriété
  - Optimal dans la classe des algorithmes à priorités fixes pour des tâches périodiques indépendantes à échéance sur requête ( $D_i=P_i$ ).

Option STR – 2006-2007

37

### Ordonnement préemptif à priorité fixe Rate Monotonic (2/2)

Tâche	Pi	Ci	Di
T1	20	3	20
T2	5	2	5
T3	10	2	10



Option STR – 2006-2007

38

### Ordonnement Parenthèse : faisabilité pour tâches périodiques

Tâche	Pi	Ci	Di
T1	20	3	20
T2	5	2	5
T3	10	2	10

- Taux d'occupation processeur (ou charge) par tâche :  $U_i = C_i / P_i$
- Taux d'occupation processeur (ou charge) pour l'ensemble des tâches :  $U = \sum C_i / P_i$
- Remarque : Pour que toutes les échéances soient respectées, il est **nécessaire** (mais **pas suffisant**) que  $U \leq 1$  (Conditions de faisabilité détaillées à la fin du cours)
- Exercice : calculer les charges (par tâche, globale) pour l'ensemble de tâches donné en exemple

Option STR – 2006-2007

39

### Ordonnement préemptif à priorité fixe Deadline Monotonic (1/2)

- Pour tâches périodiques
- Définition [Leung & Whitehead, 1985]
  - La priorité d'une tâche est inversement proportionnelle à son échéance relative (conflits résolus arbitrairement)
- Propriété
  - Optimal dans la classe des algorithmes à priorités fixes pour des tâches périodiques indépendantes à échéance  $\leq$  période
- Remarque: RM est un cas particulier de DM

Option STR – 2006-2007

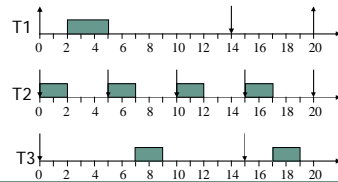
40

### Ordonnement préemptif à priorité fixe Deadline Monotonic (2/2)

- Exemple : ordonnement DM pour un jeu de tâches synchrone

Tâche	Pi	Ci	Di
T1	20	3	14
T2	5	2	5
T3	15	2	15

- $prio(T2) > prio(T1) > prio(T3)$



Option STR – 2006-2007

41

### Ordonnement préemptif à priorité dynamique Earliest Deadline First (1/3)

- Earliest Deadline First (EDF)
- Applicable pour tâches périodiques **et non périodiques**
- Définition [Liu & Layland, 1973]
  - A un instant donné, la tâche la plus prioritaire (parmi les tâches prêtes) est celle dont **l'échéance absolue est la plus proche**
  - Préemptif
- Propriété
  - Optimal dans la classe des algorithmes préemptifs pour des configurations de tâches périodiques indépendantes avec échéance  $\leq$  période

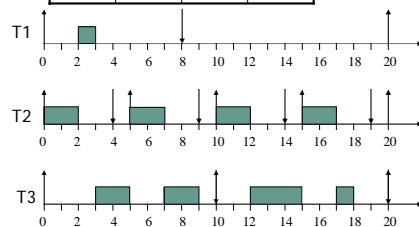
Option STR – 2006-2007

42

### Ordonnement préemptif à priorité dynamique Earliest Deadline First (2/3)

Tâche	Pi	Ci	Di
T1	20	1	8
T2	5	2	4
T3	10	4	10

(2 préemptions en 5 et 15)



Option STR – 2006-2007

43

### Ordonnement préemptif à priorité dynamique Earliest Deadline First (3/3)

- Exercice
  - Cet ensemble de tâches serait-il faisable en utilisant un ordonnement non préemptif ?
  - Serait-il faisable en utilisant un ordonnement préemptif à priorités fixes (si oui, lequel ?)

Tâche	Pi	Ci	Di
T1	20	1	8
T2	5	2	4
T3	10	4	10

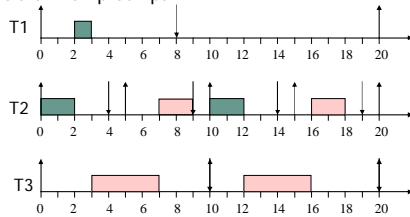
Option STR – 2006-2007

44

Exercice  
Correction (1/3)

Tâche	Pi	Ci	Di
T1	20	1	8
T2	5	2	4
T3	10	4	10

- Possible en non préemptif



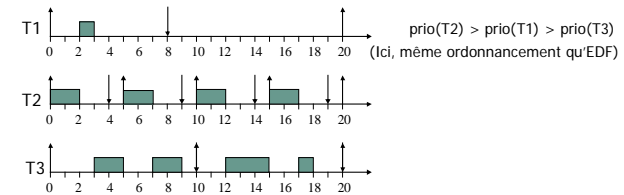
Option STR – 2006-2007

45

Exercice  
Correction (2/3)

Tâche	Pi	Ci	Di
T1	20	1	8
T2	5	2	4
T3	10	4	10

- Priorités fixes : résultat d'optimalité de DM : Si faisable, l'est nécessairement sous DM. Ici, possible.



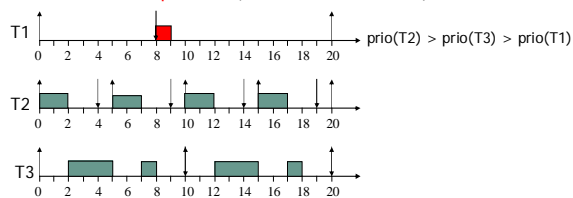
Option STR – 2006-2007

46

Exercice  
Correction (3/3)

Tâche	Pi	Ci	Di
T1	20	1	8
T2	5	2	4
T3	10	4	10

- Essai avec RM. Pas possible (T1 rate son échéance)



Option STR – 2006-2007

47

Ordonnement préemptif à priorité dynamique  
Least Laxity First (LLF) (1/2)

- Least Laxity First (LLF)
- Définition [Mok & Dertouzos, 1989]
  - A un instant donné, la tâche la plus prioritaire (parmi les tâches prêtes) est celle dont la **laxité**  $xl(t) = Di - (t + Ci - ci(t))$  est la **plus petite**
- Optimal dans la classe des algorithmes préemptifs pour des configurations de tâches périodiques indépendantes avec échéance  $\leq$  période

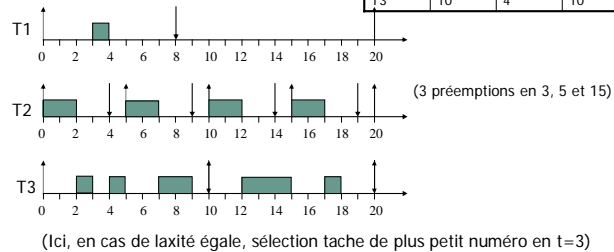
Option STR – 2006-2007

48

## Ordonnement préemptif à priorité dynamique Least Laxity First (LLF) (2/2)

- Exemple (même exemple que pour EDF)

Tâche	Pi	Ci	Di
T1	20	1	8
T2	5	2	4
T3	10	4	10



Option STR – 2006-2007

49

## Critères de choix d'un ordonnancement

- Faisabilité
  - Certaines applications nécessitent des préemptions pour respecter les contraintes de temps
- Conditions de faisabilité
  - Existent pour l'ensemble des ordonnancements présentés
  - Borne de charge atteignable plus importante pour RM/DM que pour EDF
 
$$\sum_{i=1}^n \frac{C_i}{P_i} \leq n(2^{\frac{1}{n}} - 1)$$

$$\sum_{i=1}^n \frac{C_i}{P_i} \leq 1$$
  - Mais mauvaise tolérance aux surcharges d'EDF (effet domino)
- Mise en œuvre
  - Ordonnement hors-ligne: simple (pas besoin d'un OS)
  - Ordonnement en-ligne: priorités statiques plus faciles à mettre en œuvre
- Des aspects différents entrent en jeu ...

Option STR – 2006-2007

50

## Exécutifs temps-réel multitâches

## Pourquoi utiliser des systèmes multi-tâches ?

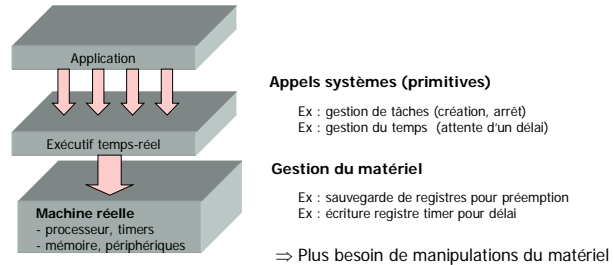
- Objectif d'un système multitâche
  - Faire coexister plusieurs traitements (calculs) sur le même processeur
- Pourquoi utiliser un système multitâche ?
  - Mieux exploiter le processeur pendant les entrées-sorties
    - Attente active : occupation processeur pendant l'E/S (sous-exploité)
    - Pas d'attente (polling, gestion sous IT) : libération du processeur
  - Parallélisme intrinsèque des applications
    - Fonctions à mettre en œuvre spécifiées indépendamment
    - Moins de processeurs que de fonctions  $\Rightarrow$  **partage** du processeur, **ordonnement** des fonctions
    - Certains jeux de tâches respectent leurs échéances uniquement en préemptif : cf exemple introductif

Option STR – 2006-2007

52

## Rôle d'un exécutif temps-réel

- ❑ Objectif : masquer à l'application les particularités du matériel, interface entre l'application et le matériel
  - **Machine virtuelle** plus ou moins complexe



Option STR – 2006-2007

53

## Types de services fournis

- ❑ Gestion des **tâches**
  - Création, destruction, activation, arrêt, changement de priorité
- ❑ Gestion des **synchronisations** (précédences + partage de ressources)
  - Création objets de synchronisation (ex: sémaphores, signaux), attente, réveil
- ❑ Gestion des **communications**
  - Création objets de communication (ex: boîtes à lettres), envoi, réception
- ❑ Gestion de la **mémoire**
  - Allocation (statique/dynamique) de zones de mémoire partagée
- ❑ Gestion du **temps**
  - Attente d'un délai, activation périodique de tâches
- ❑ Gestion des **périphériques**

Option STR – 2006-2007

54

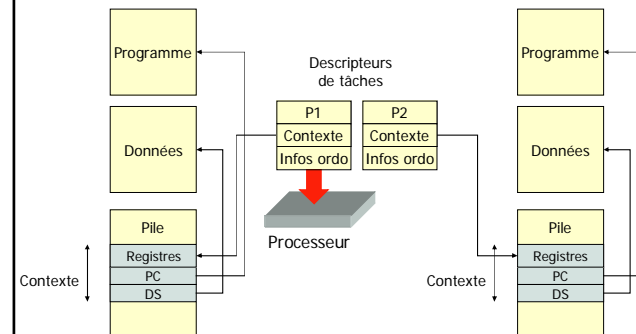
## Gestion des tâches Terminologie (1/2)

- ❑ **Programme**
  - Entité constituée des instructions à dérouler pour effectuer un calcul (statique)
- ❑ **Processus, tâche, thread**
  - Entité dynamique composée de :
    - Programme (composante statique)
    - Données, pile d'exécution, contexte (composante dynamique)
  - **Descripteur** : structure de donnée regroupant les informations sur une tâche à un instant donné (nom, contexte d'exécution, ...)
  - Thread vs tâche/processus : pas d'espace d'adressage séparé pour les threads
- ❑ Exemple : lecture périodique d'un capteur de température
  - Un programme (exécutable)
  - Une tâche relancée périodiquement

Option STR – 2006-2007

55

## Gestion de tâches Terminologie (2/2)



Option STR – 2006-2007

56

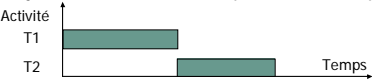

## Gestion de tâches Niveaux de parallélisme (1/2)

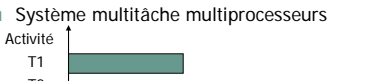
- ❑ Parallélisme réel
  - Systèmes multiprocesseurs uniquement
- ❑ Pseudo-parallélisme : le processeur exécute successivement plusieurs traitements
  - Traitements exécutés en séquence : ordonnancement **non préemptif**
  - Un traitement peut être interrompu par un autre traitement : ordonnancement **préemptif**
- ❑ Suite du cours : systèmes **monoprocesseurs**

Option STR – 2006-2007

57

## Gestion de tâches Niveaux de parallélisme (2/2)

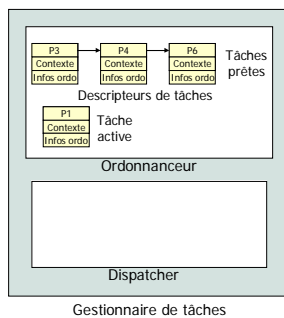
- ❑ Système multitâche monoprocesseur non préemptif
 
- ❑ Système multitâche monoprocesseur préemptif
 

Entrelacement entre T1 et T2 dépend du type d'ordonnancement
- ❑ Système multitâche multiprocesseurs
 

Option STR – 2006-2007

58

## Gestion de tâches Structure de l'exécutif

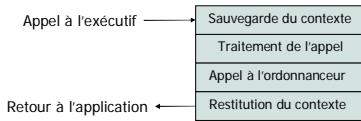


- ❑ Ordonnanceur : **choix de la tâche à exécuter** selon une politique d'ordonnancement
  - structure de données dépend de la politique d'ordonnancement (ex: tableau de files / priorités)
  - ex: tâche la plus prioritaire
- ❑ Dispatcher : allocation du processeur à la tâche sélectionnée par l'ordonnanceur (**gestion du contexte d'exécution**)
  - Sauvegarde contexte (pile)
  - Restauration contexte depuis le descripteur de tâche

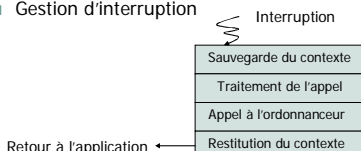
Option STR – 2006-2007

59

## Gestion de tâches Appels système et interruptions

- ❑ Appel système
 

**Module du noyau concerné**

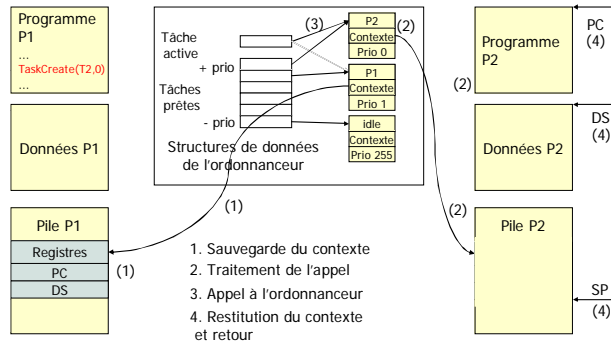
  - (dispatcher)
  - (un des modules du noyau)
  - (ordonnanceur)
  - (dispatcher)
- ❑ Gestion d'interruption
 

On ne retourne pas nécessairement dans la tâche appelante/interrompue

Option STR – 2006-2007

60

## Gestion de tâches Exemple de déroulement d'un appel système

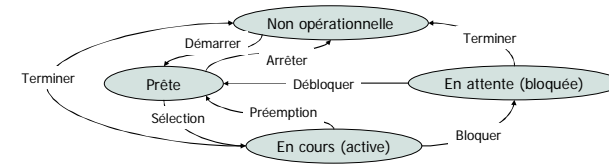


Option STR – 2006-2007

61

## Gestion de tâches Etats d'une tâche

- Partage du processeur par plusieurs tâches
  - Notions de tâche **prête** et tâche **active**
- Primitives de synchronisation : **blocage** des tâches (⇒ état **bloquée**)
- Diagramme de transition entre états



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

Option STR – 2006-2007

62

## Gestion de tâches Types d'ordonnanceurs

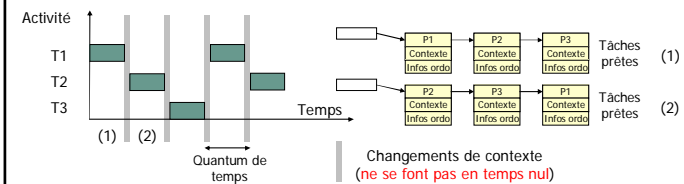
- Ordonnanceurs non préemptifs (séquenceurs)
  - Jouent une séquence d'exécution générée statiquement (non préemptif)
  - Noyaux propriétaires essentiellement (non commerciaux) + spécification OSEKtime du consortium OSEK/VDX
- Ordonnanceurs préemptifs
  - Ordonnanceurs **à priorités**
    - Priorité par tâche, allocation du processeur à la tâche la plus prioritaire (préemptif)
    - Quasi-totalité des exécutifs temps-réel du marché
  - De type **temps-partagé**
    - Allocation du processeur par tranche de temps
    - Extensions au temps-réel de systèmes d'exploitation généralistes, ou pour ordonnancement des tâches de priorités identiques

Option STR – 2006-2007

63

## Gestion de tâches Ordonnanceurs temps-partagé (1/2)

- Principe
  - Allocation du processeur par tranche (quantum) de temps (time slice)



- Intérêts
  - Equité de l'attribution du processeurs entre les tâches

Option STR – 2006-2007

64

## Gestion de tâches Ordonnanceurs temps-partagé (2/2)

### Limitations

- Pas de prise en compte des importances relatives des tâches
- Difficulté de choix de la tranche de temps
  - si trop grand, augmente les temps de réponse
  - si trop petit, trop de surcoût dus aux changements de contexte
- Peu de résultat pour vérifier a priori le respect des échéances (surtout quand utilisé conjointement avec ordonnancement par priorités)

### Domaines d'utilisation

- Extensions au temps-réel d'exécutifs généralistes
- Exécutifs temps-réel
  - Utilisé pour le partage de temps entre tâches de même priorité
  - En option (ex: noyau wind de VxWorks)

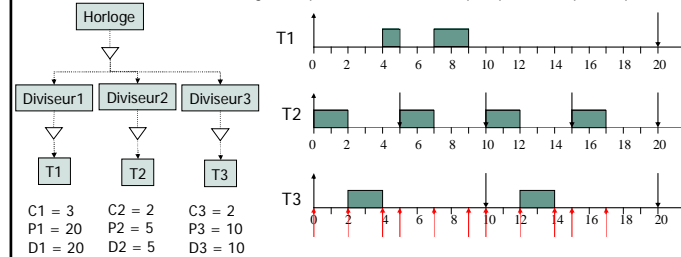
Option STR – 2006-2007

65

## Gestion de tâches Ordonnanceurs à priorités

- Assignation d'une priorité à chaque tâche
- Allocation du processeur (état = active) à la tâche prête la plus prioritaire

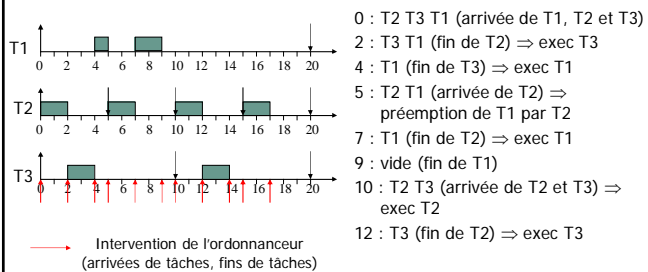
(Assignment priorités selon RM : T2 plus prioritaire que T3, puis T1)



Option STR – 2006-2007

66

## Gestion de tâches Ordonnanceurs à priorités



Option STR – 2006-2007

67

## Gestion de tâches Ordonnanceurs à priorités

### Intérêts

- Prise en compte de l'« importance » différente des tâches (assignation des priorités statiquement ou dynamiquement selon les contraintes temporelles)
- Nombreux résultats permettant de vérifier a priori les échéances (voir plus loin)

### Limitations

- Ordonnancements préemptifs : coût des changements de contexte (non nul, difficile à estimer précisément)

### Remarque

- La transformation contrainte de temps ⇒ priorité est à la charge de l'utilisateur
- Pas de notion de contrainte de temps
- Pas de vérification des contraintes de temps

Option STR – 2006-2007

68

## Gestion de tâches

### Ordonnanceurs préemptifs et mémoires cache

- ❑ Mémoires cache : mémoire à accès rapide
  - Stocke les informations (**données, instructions**) les plus souvent utilisées dans le passé, donc ayant la plus forte probabilité de réutilisation
  - Intérêt : diminue les temps d'exécution des logiciels (Ci) de manière probabiliste
- ❑ Caches ⇒ **augmentation temps de changement de contexte**
  - Changement de contexte : changement tâche exécutée (code, données)
  - Après changement de contexte : contenu du cache à réinitialiser
  - Au pire, rechargement **complet** du cache (élevé pour les gros caches)
  - Problème similaire de « pollution » du cache pour les routines de traitement d'interruptions
- ❑ Caches ⇒ **difficulté accrue d'estimation des temps d'exécution**
- ❑ Routines de gestion de cache de VxWorks (lock/unlock, validate/invalidate, flush)

Option STR – 2006-2007

69

## Gestion de tâches

### Primitives classiques

- ❑ Démarrage d'une tâche (non opérationnelle → prête)
- ❑ Arrêt d'une tâche (→ non opérationnelle)
- ❑ Suspension d'une tâche (prête → en attente)
- ❑ Réactivation d'une tâche (en attente → prête)
- ❑ Changement de priorité d'une tâche
  - Permet de mettre en œuvre des ordonnancements à assignation dynamique de priorités (**non disponible de base** dans les exécutifs temps-réel)

Option STR – 2006-2007

70

## Gestion de tâches

### Exemples de primitives

Primitive	xec_86	OSEK/VDX	VxWorks
<b>Démarrage</b>	InitTask(No,prio,adstart,adstack, ...); StartTask(No);	ActivateTask(tid);  Schedule(); /* Appel explicite à l'ordonnanceur */	taskInit(ad tcb, Nname, Prio, Opt, adstack, StSize, adstart, ...); tid = taskSpawn(Name, Prio, Opt, StSize, adstart); taskActivate(tid);
<b>Arrêt</b>	StopTask(No); Terminate();	TerminateTask(); ChainTask(tid); /* Arrêt au profit de tid */	taskDelete(Tid);
<b>Manipulation priorités</b>	TaskPriority(No); ChangePriority(No, Prio);	Aucune (priorités non modifiables)	taskPrioritySet(Tid, priority); taskPriorityGet(Tid, adpriority);
<b>Suspension / reprise</b>		Aucune	taskSuspend(Tid); taskResume(Tid);

Option STR – 2006-2007

71

## Gestion de tâches

### Du modèle fonctionnel à l'exécutif

- ❑ Fonction → tâche de l'exécutif
- ❑ Choix d'une politique d'ordonnement : sélection selon l'analyse a priori du système (voir plus loin)
  - Hors-ligne : **pas besoin d'exécutif** (un simple séquenceur suffit)
  - En-ligne
    - A priorités statiques : assignation des priorités à la création de la tâche (on n'utilise pas la primitive de gestion des priorités)
      - **Faible coût de mise en oeuvre**
    - A priorités dynamiques : assignation des priorités à la création des tâches + dynamiquement quand nécessaire
      - on utilise la primitive de modification des priorités (exemple : pour EDF, on reclasse les tâches par priorité à chaque arrivée d'une nouvelle tâche pour les stocker par échéance croissante)
      - **Coût de mise en oeuvre plus important (n log N pour tri)**

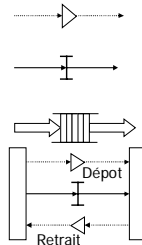
Option STR – 2006-2007

72

## Synchronisation et communication

### Relations inter-tâches de trois types

- ❑ **Précédence (synchronisation par événement)**
  - notion d'ordre entre les traitements
- ❑ **Partage de variables / ressources**
  - Echanges d'informations ou de résultats
- ❑ **Communications**
  - Précédence + transfert d'information : relation de type producteur/consommateur
  - Association des deux relations précédentes
- ❑ Primitives de synchronisation/communication offertes par les exécutifs temps-réel pour ces différents types de synchronisation/communication



Option STR – 2006-2007

73

## Synchronisation

- ❑ Rôle
  - Pour attendre un événement avant d'exécuter un calcul
- ❑ Origine des événements
  - Matériel : capteurs
  - Logiciel : synchronisation interne à une application
- ❑ Classification des primitives de synchronisation
  - Mémorisation
    - Sans mémorisation (événement fugace, prise en compte sur niveau)
      - Si tâche en attente, alors on la réveille, sinon l'évt est perdu
    - Avec mémorisation (événement persistant, prise en compte sur état)
      - sans compte : **une** occurrence est mémorisée jusqu'à sa consommation (« effacement » de l'événement peut être ou non à la charge de l'application) : **événements**
      - à compte : **compteur** d'occurrences non effacées : **sémasphores**

Option STR – 2006-2007

74

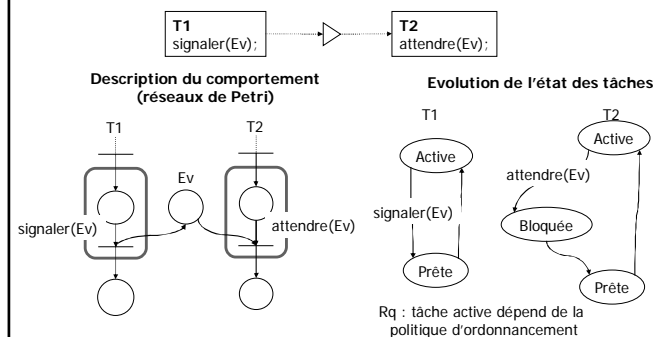
## Synchronisation

- ❑ Classification des primitives de synchronisation (suite)
  - Directe / indirecte
    - Directe : on signale à une tâche précise (signal(Ev,T);)
    - Indirecte : global (signal(Ev);)
  - Synchrone / asynchrone
    - Synchrone : la tâche se met **explicitement** en attente de l'événement (wait(Ev);)
    - Asynchrone : on associe à une tâche un gérant d'événement qui est exécuté lors de l'occurrence de l'événement (connecter\_Gérant(Ev, fonction);)
  - Public / privé
    - Privé : jeu d'événements par tâches (pas de création d'événements)
    - Public : global au système (primitive de création d'événement)

Option STR – 2006-2007

75

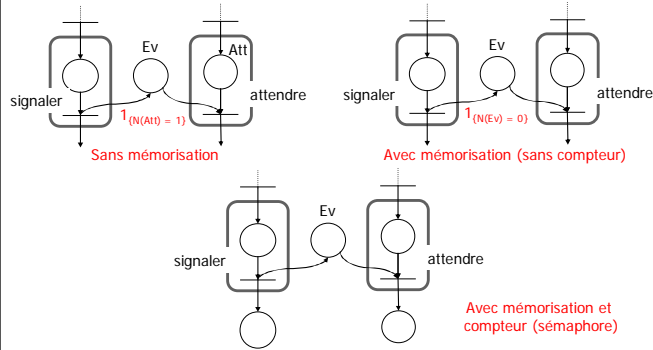
## Synchronisation



Option STR – 2006-2007

76

## Synchronisation



Option STR – 2006-2007

77

## Synchronisation par événements

### Exemple OSEK/VDX

- Types d'événements
  - Événements avec **mémorisation sans compte**, effacement **explicite**  
Événements **privés** (nombre limité d'événements par tâches, pas d'appel de création d'événement)
  - Signal **direct** des événements
  - **Synchrone** : appel explicite à la routine d'attente d'un événement
- Interface
  - SetEvent (tid,mask) : positionne l'ensemble d'événements de la tâche *tid* désignés par le masque *mask* (bitmap: 1 = set, 0 = clear)
  - ClearEvent (mask) : effacement des événements désignés par *mask*
  - WaitEvent (mask) : attente (sans délai) des événements désignés par *mask* (attente sur un ensemble d'événements)
  - GetEvent (tid,mask) : retourne les événements à 1 du masque *mask* concernant la tâche *tid* (pas de relation avec événements attendus)

Option STR – 2006-2007

78

## Synchronisation par événements Du modèle fonctionnel à l'exécutif (1/2)

### □ Modèle fonctionnel



### □ Correspondance avec les objets de l'exécutif

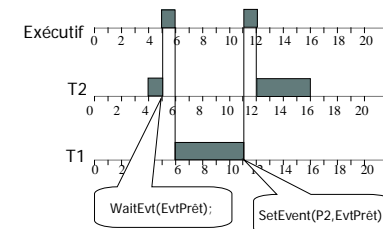
- Fonction → tâche
- Événement → objet de synchronisation approprié de l'exécutif  
(sur OSEK-VDX, *SetEvent (T2, EvtPrêt)* dans T1,  
*WaitEvent (EvtPrêt); ClearEvent(EvtPrêt);* dans T2).

Option STR – 2006-2007

79

## Synchronisation par événements Du modèle fonctionnel à l'exécutif (2/2)

### □ Chronogramme de l'exemple précédent (T2 plus prioritaire que T1)

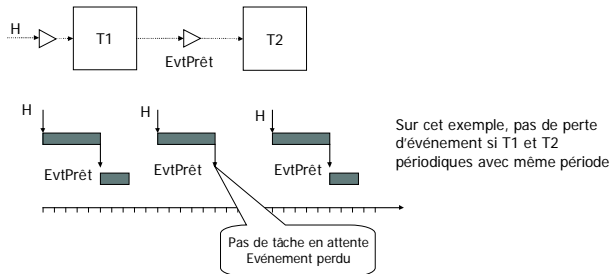


Option STR – 2006-2007

80

## Synchronisation par événements Exemple de pertes d'événements

- Événements sans mémorisation (pSOS, Sceptre)



Option STR – 2006-2007

81

## Synchronisation par événements Implantation

- Synchronisation par événement mémorisé sans compteur, effacement implicite des événements

```
typedef struct {
    int état; /* Inactif, Bloqué, Prêt */
    t_ctx contexte; /* Contexte de la tâche
} descr_tâche; /* Descripteur de tâche */
descr_tâche T[MAX_TACHES]; /* Tableau des descripteurs de tâches */
char Ev[MAX_EVT]; /* Tableau des descripteurs d'évt : Set/Clear */

Signal(int tid, int e) {
    Sauvegarde contexte;
    Ev[e] = Set;
    if (T[tid].état == Bloqué) {
        T[tid].état = Prêt;
        Ev[e] = Clear;
    }
    Appel à l'ordonnanceur;
    Restitution du contexte;
}

Wait(int e) {
    Sauvegarde contexte;
    if (Ev[e] == Set) {
        Ev[e] = Clear;
    }
    else T[Active].état = Bloqué;
    Appel à l'ordonnanceur;
    Restitution du contexte;
}
```

Pas de compteur  
un booléen suffit

Option STR – 2006-2007

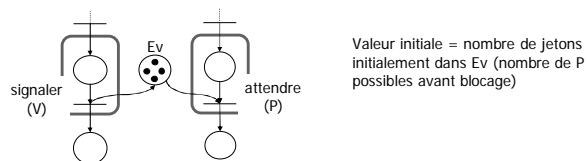
82

## Synchronisation par sémaphores Principe

- Primitives classiques

- sid = create\_sema(cpt); /\* cpt >= 0 \*/
- P (sid); /\* Puis-je ? \*/
- V (sid); /\* Vas - y \*/

- Comportement



Option STR – 2006-2007

83

## Synchronisation par sémaphores Implantation

```
typedef struct {
    int cpt; /* Compteur
    ≥ 0 = nombre de P avant blocage (nb evts non consommés)
    < 0 = nombre d'éléments dans la file d'attente */
    file *F; /* Tâches en attente sur le sémaphore */
} t_sema; /* Descripteur de sémaphore */
```

```
V (t_sema *s) {
    Sauvegarde contexte;
    (s->cpt) ++;
    if (s->cpt ≤ 0) {
        T = retirer(s->F);
        T->état = Prêt;
    }
    Appel à l'ordonnanceur;
    Restitution du contexte;
}

P (t_sema *s) {
    Sauvegarde contexte;
    (s->cpt) --;
    if (s->cpt < 0) {
        Active.état = Bloqué;
        insérer(Active,s->F);
    }
    Appel à l'ordonnanceur;
    Restitution du contexte;
}
```

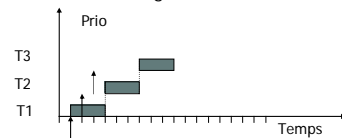
Option STR – 2006-2007

84

Synchronisation  
Impact de la gestion des files d'attente (1/2)

Illustration sur la synchronisation par sémaphores

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



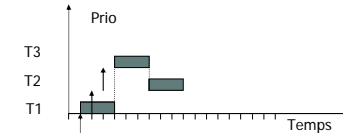
Option STR – 2006-2007

85

Synchronisation  
Impact de la gestion des files d'attente (2/2)

Illustration sur la synchronisation par sémaphores

- Cas d'une file gérée par priorité décroissante



- File gérée en FIFO : équité entre les tâches, pas de risque de famine
- File gérée par priorité : moins équitable, mais favorise les tâches plus prioritaires → plus adapté aux systèmes temps-réel
  - La gestion des files d'attente **joue un rôle important** pour le respect des échéances (xec\_86, VxWorks : choix FIFO / priorité paramétrable)
  - Applicable à tout objet de synchro nécessitant une file d'attente

Option STR – 2006-2007

86

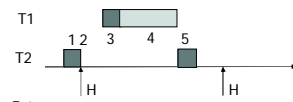
Synchronisation  
Besoin d'indivisibilité (1/2)

Exemple mettant en évidence le problème

```

T1 {
  cycle
  Wait(H);
  Signal(T2,Evt);
  Traitement_T1;
  fncycle;
}
T2 {
  cycle
  Wait(Evt);
  Traitement_T2;
  fncycle;
}
    
```

Situation initiale : Evt = Clear; T1 bloqué (sur H), T2 Active



```

Wait(int e) {
  Sauvegarde contexte;
  if (Ev[e] == Set) Ev[e] = Clear; H
  else T[Active].état = Bloqué;
  Appel à l'ordonnanceur + restitution
  contexte;
}
    
```

- T2 teste Evt et le trouve à Clear
- H arrive → préemption de T2 par T1
- T1 positionne Evt à Set et ne réveille pas T2 car T2 n'est pas (encore) bloqué
- T1 exécute son traitement
- T2 reprend, et se bloque (à cause du test fait en 1) → **T2 ne perçoit pas Evt**

⇒ Exécution **INDIVISIBLE** des primitives

Option STR – 2006-2007

87

Synchronisation  
Besoin d'indivisibilité (2/2)

- Problème d'indivisibilité
  - Valable pour **tous** les appels de synchronisation/communication (événements, sémaphores, messages)
- Mise en œuvre de l'indivisibilité des appels système
  - Invalidation ou masquage des interruptions
    - Mise en œuvre simple, peu de coût à l'exécution (cli / sti)
    - Perte d'événements possible
    - Applicable uniquement sur machine monoprocesseur (notion d'interruption locale à un processeur + bus partagé)
    - Utiliser pour des sections critiques **courtes**, en monoprocesseur
  - Instructions spécifiques du matériel (Test-and-set, exchange)

Option STR – 2006-2007

88

## Synchronisation par événements Exemples de primitives

Primitive	xec_86	OSEK / VDX	VxWorks (non exhaustif)
<b>Création / initialisation</b>	Aucune	Aucune (nb borné d'événements par tâche)	oldhandler = signal (signo, ,handler);
<b>Signal</b>	SignalEvent(NumTask, NumEvent);	SetEvent(tid, masque);	kill(tid, signo); raise(signo); /* Auto-signal */
<b>Attente / effacement / consultation</b>	WaitEvents(Mask, Timeout, &MaskOccurred); ClearEvents(Mask); EventsOccurred(&Mask);	WaitEvent(masque); ClearEvent(masque); GetEvent(tid, &masque);	sigsuspend(masque); sigtimedwait (masque, info, timeout);
<b>Destruction</b>	Aucune	Aucune	

Option STR – 2006-2007

89

## Synchronisation par sémaphores Exemples de primitives

Primitive	xec_86	OSEK / VDX	VxWorks
<b>Initialisation</b>	InitSemaphore(NoSem, valeurMax);	Pas de sémaphore	sid = semCCreate(Opt, Initialcount);
<b>Destruction</b>	-		semDelete(sid);
<b>Acquisition (« Wait »)</b>	P(NoSem, Tout); PWithPrio(NoSem, Tout);		semTake(semid, timeout);
<b>Libération (« signal »)</b>	V(NoSem);		semGive(semid); semFlush(semid); /* Libère toutes les tâches en attente */
<b>Consultation</b>	TestP(NoSem);		semShow(semid, level);

Option STR – 2006-2007

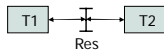
90

## Partage de ressources

- Variable partagée → deux tâches ne doivent pas y accéder en lecture et écriture simultanément

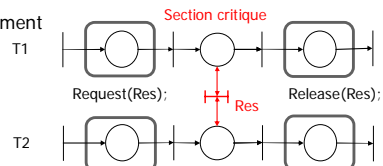
- Notion de **section critique**, accès en **exclusion mutuelle**

- Modèle fonctionnel



- Rq 1: Res peut être une ressource matérielle (capteur) ou logicielle (tampon en mémoire, fichier)
- Rq 2: Si Res est une zone de mémoire, l'exécutif doit permettre le partage de mémoire (cf plus loin)

- Comportement



Option STR – 2006-2007

91

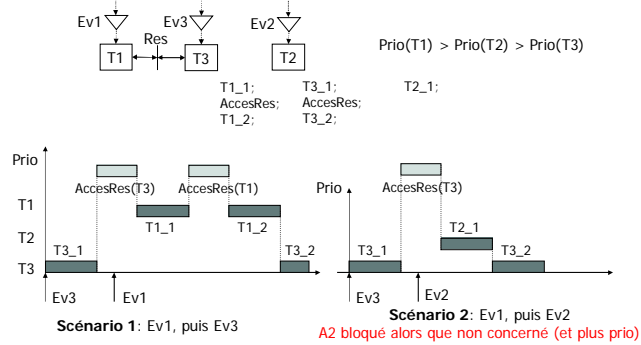
## Partage de ressources Implantation d'une section critique (1/6)

- Suppression/masquage des interruptions
  - Faible coût à l'exécution (cli/sti)
  - Latence (gigue) dans le traitement des interruptions, voire perte d'interruptions
  - Non valable en multiprocesseurs
  - Utilisable uniquement sur des sections critiques **courtes**, de **taille bornée**
- Augmentation temporaire de la priorité pendant la section critique
  - Retard dans le traitement des tâches non concernées par la ressource

Option STR – 2006-2007

92

Partage de ressources  
Implantation d'une section critique (2/6)



Option STR – 2006-2007

93

Partage de ressources  
Implantation d'une section critique (3/6)

❑ Implantation par attente active (pas vraiment une bonne idée !)

```
void Request (char *V)          void Release (char *V)
{
    while (*V) ; /* Attente ressource */
    *V = 1;
}
{
    *V = 0; /* Libération */
}
```

- Condition de correction
  - Assurer l'indivisibilité
  - mêmes méthodes que pour la synchronisation par événements
- Problème
  - Monopolisation du processeur
  - Situation de famine quand une tâche prioritaire attend une ressource possédée par une tâche moins prioritaire : **ne fonctionne pas !**

Option STR – 2006-2007

94

Partage de ressources  
Implantation d'une section critique (4/6)

❑ Utilisation de sémaphores (attente passive)

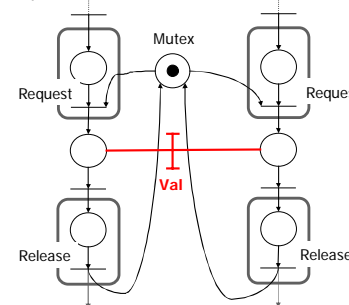
- Request(v) → P(s\_v);  
Release(v) → V(s\_v);
- Sémaphores à **compteurs** tels que ceux présentés avant
- Sémaphores d'**exclusion mutuelle (mutex)** : cas particuliers de sémaphores avec en plus les propriétés suivantes :
  - Valeur initiale de 1 (pas de valeur en paramètre de la création)
  - Restriction des appels système pour que le Request (P) et le Release (V) soient faits par la même tâche (ex: VxWorks)
  - Possibilité d'emboîter dans la même tâche (sans blocage) des accès à la même ressource (ex: VxWorks)

Option STR – 2006-2007

95

Partage de ressources  
Implantation d'une section critique (5/6)

❑ Comportement des mutex

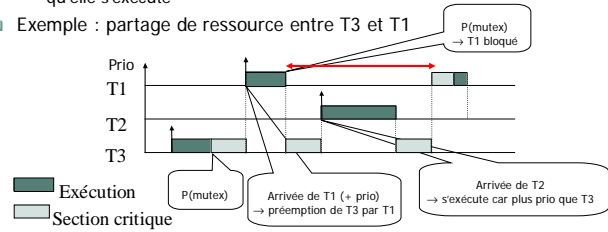


Option STR – 2006-2007

96

Partage de ressources  
Problème d'inversion de priorité

- Définition de l'inversion de priorité
  - une tâche prioritaire est bloquée alors qu'une tâche moins prioritaire qu'elle s'exécute
- Exemple : partage de ressource entre T3 et T1



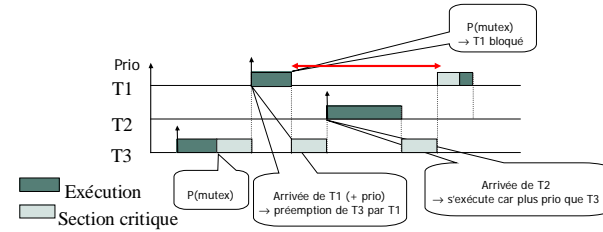
T1 bloqué pendant toute la durée de T2. Durée blocage potentiellement infinie (multiples tâches de prio. interm.).

Option STR – 2006-2007

97

Partage de ressources  
Problème d'inversion de priorité

- On voit sur l'exemple précédent qu'une gestion des files d'attente par priorité n'élimine pas l'inversion de priorité



Option STR – 2006-2007

98

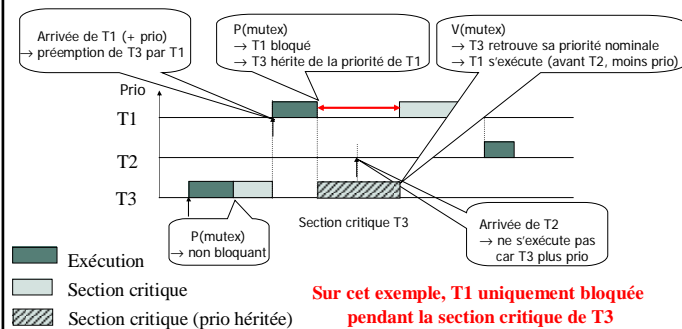
Partage de ressources  
Solution à l'inversion de priorité : PIP (1/9)

- Protocole d'héritage de priorité (PIP = Priority Inheritance Protocol)
  - Sha, Rajkumar, Lehoczky, 90
- Prérequis
  - Ordonnement à priorité fixe, priorité nominale vs priorité courante
  - Mutex pour sections critiques, section critiques "properly nested"
- Définition du protocole
  1. Règle d'ordonnement: Exécution selon priorité nominale, sauf règle 3
  2. Règle d'allocation de ressource
    - a) Ressource libre : acquisition de la ressource
    - b) Ressource occupée : blocage
  3. Règle d'héritage de priorité : qd blocage d'une tâche T lors de l'accès à R, la tâche bloquante T1 hérite de la priorité courante de T; utilisation par T1 de la priorité héritée jusqu'à libération de R, où elle retrouve la priorité de la tâche la plus prioritaire bloquée sur R.

Option STR – 2006-2007

99

Partage de ressources  
Solution à l'inversion de priorité : PIP (2/9)



Sur cet exemple, T1 uniquement bloquée pendant la section critique de T3

Option STR – 2006-2007

100

Partage de ressources  
Solution à l'inversion de priorité : PIP (3/9)

- Propriétés
  - Temps de blocage borné (sauf situation d'interblocage)
    - Possibilité de vérification a priori des échéances
  - Interblocages possibles
  - Pour implanter PIP, il n'est pas nécessaire de connaître quelle tâche utilise quelle ressource (par contre, nécessaire pour calculer les temps de blocage)
- Remarque
  - N'élimine pas l'inversion de priorité, la rend juste de durée bornée

Option STR – 2006-2007

101

Partage de ressources  
Solution à l'inversion de priorité : PIP (4/9)

- Remarque: héritage transitif en cas de sections critiques emboîtées
  - si T partage une ressource R avec T' moins prio qu'elle
  - si la section critique sur R inclut l'utilisation d'une ressource R'
  - si R' est utilisée par une tâche T'' moins prioritaire que T'
  - T' dépend de T'' et par transitivité, T dépend de T''
- Temps de blocage
  - Définition : durée pendant laquelle une tâche T ne peut plus progresser à cause d'une tâche moins prioritaire qu'elle
  - Remarque : une tâche prête peut subir un « blocage » tel qu'il est défini ici

Option STR – 2006-2007

102

Partage de ressources  
Solution à l'inversion de priorité : PIP (5/9)

- Sources de blocage
  - Blocage direct : quand T tente d'accéder une ressource utilisée par une tâche moins prioritaire qu'elle
    - Subi par tâche prioritaire partageant une ressource avec une tâche moins prioritaire
  - Blocage indirect (push-through blocking) : quand T est bloqué par une tâche moins prioritaire qui a hérité d'une priorité plus importante que T via le mécanisme d'héritage de priorité
    - Subi par tâches de priorité intermédiaire ne partageant pas nécessairement de ressources



Option STR – 2006-2007

103

Partage de ressources  
PIP : Calcul de temps de blocage (6/9)

- Algorithme fonctionnant pour les sections critiques non emboîtées seulement
- Complexité  $O(m \cdot n^2)$
- Variante plus précise (borne plus faible) de complexité exponentielle [Rajkumar, 91]

Option STR – 2006-2007

104

Partage de ressources  
PIP : Calcul de temps de blocage (7/9)

- Algorithme de calcul du blocage maximal subi par  $T_i$
- 1. Pour chaque tâche  $T_j$  de priorité inférieure à  $T_i$ 
  - Identifier les sections critiques de  $T_j$  pouvant bloquer  $T_i$  et prendre la plus longue
  - Sommer les durées des  $n$  sections critiques ainsi identifiées pour l'ensemble des  $T_j$
- 2. Pour chaque sémaphore d'exclusion mutuelle  $sm$ 
  - Identifier les  $m$  sections critiques protégées par  $sm$  qui peuvent bloquer  $T_i$  et prendre la section critique la plus longue
  - Sommer les durées des  $m$  sections critiques correspondant aux durées ainsi sélectionnées
- Prendre le minimum entre les deux valeurs

Option STR – 2006-2007

105

Partage de ressources  
Solution à l'inversion de priorité : PIP (8/9)

- Calcul du temps de blocage sur l'exemple (tâche  $T_1$ )
  - Item 1
    - Tâche  $T_2$ 
      - pas de blocage direct (pas de partage de ress. entre  $T_1$  et  $T_2$ )
      - pas de push-through (ni  $T_2$  ni  $T_3$  ne peuvent hériter la priorité de tâches plus prioritaires que  $T_1$ )
    - Tâche  $T_3$ 
      - blocage direct (partage de  $R$  avec  $T_3$ ) :  $n=1$
      - pas de push-through blocking
  - Item 2 (examen de tous les sémaphores pouvant bloquer  $T_1$ , idf  $m$ )
    - un seul sémaphore utilisé (pour protéger  $R$ ). Il peut provoquer uniquement un blocage direct sur  $T_1$ .  $m=1$
- $\min(n,m)$  = la section critique protégée par  $R$

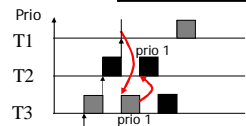
Option STR – 2006-2007

106

Partage de ressources  
Solution à l'inversion de priorité : PIP (9/9)

- Remarque
  - d'après la définition des temps de blocage, la tâche la moins prioritaire aura toujours un temps de blocage maximum nul
- Exemple

Tâche	Arrivée $A_i$	WCET $C_i$	prio( $T_i$ ) (1 = + prio)	Ressources critiques
T1	2	1	1	[Gris:1]
T2	1	2	2	[Noir:2]
T3	0	3	3	[Gris:2 [Noir:1]]



Temps de blocage de  $T_3$  nul,  
Blocage de  $T_3$  à la date 3 non comptabilisé  
dans son temps de blocage maximal

Option STR – 2006-2007

107

Partage de ressources  
Solution à l'inversion de priorité : PCP (1/6)

- PCP = Priority Ceiling Protocol (Protocole à priorité « plafond »)
  - ex: OSEK/VDX
- Prérequis
  - Ordonnancement à priorité fixe
  - Ressources utilisées par toutes les tâches sont connues a priori (contrairement à PIP)
- Définitions
  - **Priorité plafond** (priority ceiling) d'une ressource  $\Pi(R)$  = priorité maximale des tâches qui l'utilisent
  - **Priorité plafond courante** (current priority ceiling, ou ceiling) à un instant  $t$  :  $\Pi(t)$  = maximum des priorités plafond des ressources **utilisées** en  $t$

Option STR – 2006-2007

108

Partage de ressources  
Solution à l'inversion de priorité : PCP (2/6)

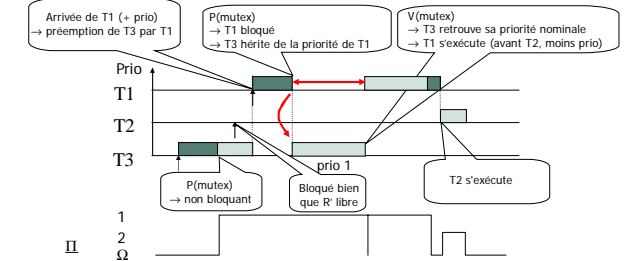
- Protocole
  1. Règle d'ordonnement: Exécution selon priorité nominale, sauf règle 3 (idem PIP)
  2. Règle d'allocation de la ressource R demandée par une tâche T
    - a) Ressource occupée : blocage (idem PIP)
    - b) Ressource libre
      - i. T strictement plus prioritaire que  $\Pi(t)$ , allocation de R à T
      - ii. Sinon, R allouée à T si c'est T qui possède la ressource de plafond  $\Pi(t)$ , sinon, T est bloquée (différent PIP)
  3. Règle d'héritage de priorité : qd blocage d'une tâche T lors de l'accès à R, la tâche bloquante TI hérite de la priorité de T  $prio(t)$ , et ce jusqu'à ce qu'il libère toute ressource de plafond supérieur ou égal à  $prio(t)$ . A ce moment, TI reprend la priorité qu'il avait à quand il a acquis la ressource.

Option STR – 2006-2007

109

Partage de ressources  
Solution à l'inversion de priorité : PCP (3/6)

- Exemple : partage de ressource R entre T1, T3, T2 utilise une ressource R'
  - $\Pi(R) = prio(T1)$ ;  $\Pi(R') = prio(T2)$ ; Initialement  $\Pi(0) = \Omega$ ;



Option STR – 2006-2007

110

Partage de ressources  
Solution à l'inversion de priorité : PCP (4/6)

- Propriétés
  - **Durée de blocage bornée** : borne = durée maximale d'une section critique d'une tâche de priorité inférieure (plus faible qu'avec PIP)
  - Durée de blocage pour une tâche  $T_i$  = Durée de la plus longue section critique d'une tâche de priorité inférieure et gardée par un sémaphore s tel que  $\Pi(s) \geq prio(T_i)$ 
    - Identifier les sections critiques des tâches de priorité inférieure qui peuvent bloquer  $T_i$  (direct + push-through)
    - Filtrer toutes les sections critiques vérifiant  $\Pi(s) \geq prio(T_i)$  et prendre le maximum
  - Durée de blocage maximal  $B_i$  plus faible qu'avec PIP
  - Pas d'interblocages possibles
- Mais nécessite de connaître toutes les ressources partagées par toutes les tâches (non transparent pour l'utilisateur)
- N'élimine pas l'inversion de priorité, en rend la durée bornée

Option STR – 2006-2007

111

Partage de ressources  
Solution à l'inversion de priorité : PCP (5/6)

- Exemple de calcul des temps de blocage

	S1 ( $\Pi = T1$ )	S2 ( $\Pi = T1$ )	S3 ( $\Pi = T2$ )
T1	1	2	0
T2	0	9	3
T3	8	7	0
T4	6	5	4

- T1 : SC pouvant bloquer T1
  - pas de push-through blocking (T1 est la plus prioritaire)
  - blocage direct : T2(9), T3(8), T3(7), T4(6), T4(5)
  - après filtrage, toutes les SC répondent à la condition
  - D'ou  $B1 = \max(8,6,9,7,5) = 9$

Option STR – 2006-2007

112

Partage de ressources  
Solution à l'inversion de priorité : PCP (6/6)

Exemple de calcul des temps de blocage

	S1 (π = T1)	S2 (π = T1)	S3 (π = T2)
T1	1	2	0
T2	0	9	3
T3	8	7	0
T4	6	5	4

- T2 : SC pouvant bloquer T2
  - blocage direct : T3(7), T4(5), T4(4), push-through : T3(8), T3(6)
  - après filtrage, toutes les SC répondent à la condition
  - D'où  $B2 = \max(8, 6, 7, 5, 4) = 8$
- T3 : SC pouvant bloquer T3
  - blocage direct : T4(6), T4(6), push-through : T4(4)
  - après filtrage, toutes les SC sont conservées, D'où  $B3 = \max(6, 5, 4) = 6$ ;
- T4 :  $B4 = 0$ ;

Option STR – 2006-2007

113

Partage de ressources  
Exemples de primitives de gestion des mutex

Primitive	xec_86	OSEK / VDX	VxWorks
Initialisation	Aucune	Aucune (allocation statique)	sid = semBCreate(options, initialState); sid = semMCreate(options);
Destruction	-	Aucune	
Acquisition	EnterRegion();	GetResource(resid);	semTake(sid, timeout); taskLock(); /* Disable context switch */ intLock();
Libération	LeaveRegion();	SetResource(resid);	semGive(sid); taskUnlock(); /* Unable context switch */ intUnlock();

PCP (ceiling fixé à l'init)

PIP et gestion de files  
(FIFO, prio. configurables)

Option STR – 2006-2007

114

Gestion des communications  
(en monoprocesseur)

Modèle fonctionnel



Classification des systèmes de communication par messages

- synchrone / asynchrone
  - synchrone : la tâche se met explicitement en attente du message
  - asynchrone : gérant de message appelé à son arrivée
- avec / sans mise en file
  - avec : mémorisation dans une file de messages (boîte à lettres)
    - file de bornée / non bornée
    - gestion de file FIFO / avec priorité
  - sans : écrasement du message précédent (« tableau noir »)
- avec / sans blocage (à l'émission / réception)
  - sans blocage à l'émission → perte possible de messages si file de messages bornée

Option STR – 2006-2007

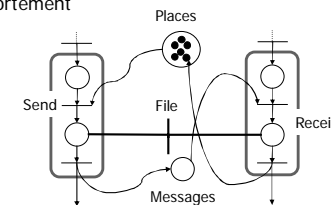
115

Gestion des communications  
Boîte à lettres, synchrone, file de taille bornée, avec blocage

Interface typique :

- Send(message, tâche);
- Receive(message);

Comportement



Option STR – 2006-2007

116

### Gestion des communications

Boîte à lettres, synchrone, file de taille bornée, avec blocage

- Implantation : problèmes à résoudre
  - Synchronisation en émission et en réception
    - blocage de l'émetteur en attente de places
    - blocage du récepteur en attente de messages
  - Exclusion mutuelle
    - Protection de la zone de données de la file de messages

Option STR – 2006-2007

117

### Gestion des communications

Boîte à lettres, synchrone, file de taille bornée, avec blocage

- Implantation à l'aide de sémaphores
  - Structure de données

```
typedef char t_msg[MAX_MSG]
typedef struct {
    t_sema s_messages init 0; /* Sémaphore pour accès aux messages */
    t_sema s_places init N_PLACES; /* Sémaphore pour accès place libre */
    t_msg tampon[N_PLACES]; /* Tampon pour message en attente */
    t_sema s_tampon init 1; /* Mutex sur tampon */
    int i_msg init 0; /* Message à lire */
    int i_place init 0; /* Place à remplir */
} t_port;
```

Option STR – 2006-2007

118

### Gestion des communications

Boîte à lettres, synchrone, file de taille bornée, avec blocage

```
void send (t_msg msg, t_port *port)
{
    /* On attend qu'une place se libère */
    P(port->s_places);

    /* Recopie message */
    P(port->s_tampon);
    recopier(&msg, port->tampon[port->i_places];
    port->i_places = (port->i_places + 1) modulo
    N_PLACES;
    V(port->s_tampon);

    /* On signale son arrivée */
    V(port->s_messages);
}

void receive (t_msg *msg, t_port *port)
{
    /* On attend qu'un message soit là */
    P(port->s_messages);

    /* Recopie message */
    P(port->s_tampon);
    recopier(port->tampon[port->i_msg], msg);
    port->i_msg = (port->i_msg + 1) modulo
    N_PLACES;
    V(port->s_tampon);

    /* On signale la libération d'une place
    vide */
    V(port->s_places);
}
```

Option STR – 2006-2007

119

### Communications

Exemples de gestion des boîtes à lettres

Primitive	xec_86	OSEK / VDX	VxWorks
<b>Initialisation / création</b>	InitMailBox(NumMbox, NumMaxMsg);	InitMessage(mid, add);	mid = msgQCreate(maxMsgs, maxMsgLength, options);
<b>Destruction</b>	Aucune	Aucune	msgQDelete (mid);
<b>Envoi</b>	Send(NumMbox, Msg); SendWithPriority(NumMbox, msg, Prio);	SendMessage(mid, add); SendDynamicMessage(mid, add, length); SendZeroMessage(mid, add);	msgQSend (mid, buffer, nBytes, timeout, priority);
<b>Réception</b>	Receive(NumMbox, &msg, Timeout);	ReceiveMessage(mid, add); ReceiveDynamicMessage(mid, add, length);	msgQReceive(mid, buffer, maxNBytes, timeout);
<b>Consultation</b>	TestReceive(NumMbox, &msg);	GetStatus(mid);	nmsg = msgQNumMsgs(mid);

Option STR – 2006-2007

120

## Gestion du temps Services classiques

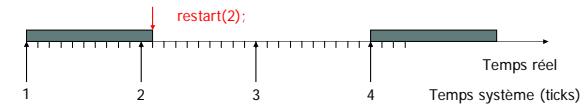
- ❑ Lecture / mise à jour du temps
  - Par **matériel** : horloge temps-réel (précision dépend du matériel)
  - Par **logiciel** : horloge système incrémentée périodiquement par routine de traitement d'interruption d'horloge (tick)
    - Granularité peut être importante (par défaut, généralement 10ms)
    - Certains systèmes permettent de changer la granularité (attention au temps passé en traitement IT horloge)
- ❑ Actions à des dates absolues
  - Différents types d'action (signal d'un événement, lancement tâche)
  - Différentes récurrences : une seule fois ou périodiquement (par exemple lancement de tâches périodiquement)

Option STR – 2006-2007

121

## Gestion du temps Services classiques

- ❑ Actions à des dates **relatives au temps courant**
  - **Chiens de garde** lors des primitives bloquantes (synchronisation, communications)
  - **Signal événement / lancement de tâche** après un certain délai
  - Attention aux relancements périodiques de tâches avec **délais relatifs** : risque de dérive par rapport au temps réel.  
Exemple : tick de 10 ms, tâche de 11 ms relancée tous les 2 ticks, relance en fin de tâche



Cause du problème : temps d'exécution de la tâche > tick (même problème avec les traitements d'interruption)

Option STR – 2006-2007

122

## Gestion du temps Services classiques

- ❑ Fonctions d'instrumentation de l'exécution
  - Exemple VxWorks
    - `timex(fn,args)`; mesure du temps pour exécuter une fonction
    - `timexN(fn,args)`; mesure du temps pour exécuter N occurrences de la fonction avec exécutions en boucle
  - Pièges à éviter
    - Mesure sans boucle : attention à la granularité de l'horloge (si temps très inférieur au tick, donnera aléatoirement 0 ou 1 tick)
    - Attention aux tests en boucle : si l'architecture est munie d'un cache, les temps peuvent être très optimistes !  

██████████	████	████	████	████	████	Test en boucle : 10 ticks
1ère exec : 100 ticks	execs suivantes :	10 ticks				
(cache quasi vide)	(cache « chaud »)					
    - Test en contexte normal : 100 cycles : le test en boucle de résout pas tout  
Se placer dans le contexte d'exécution réel de la fonction

Option STR – 2006-2007

123

## Gestion du temps Exemple des alarmes OSEK / VDX

- ❑ Alarme = association entre
  - Un compteur (ex: tick d'horloge)
  - Une tâche
  - Une action sur la tâche (**activation** / envoi d'un **événement**).
  - Attributs de l'alarme (tâche, action) fixés statiquement (init. noyau)
- ❑ Types de déclenchement de l'alarme
  - **cyclique** (récurrent) ou **unique**
  - valeur **relative** ou **absolue** du compteur (pour le premier déclenchement seulement dans le cas d'alarmes cycliques)
- ❑ Interface
  - `SetRelAlarm(id_alarm,incr,cycle)`; /\* cycle=0 : déclenchement unique \*/
  - `SetAbsAlarm(id_alarm,start,cycle)`;
  - `CancelAlarm(id_alarm)`;
  - `GetAlarm(id_alarm,&tick)`;

Option STR – 2006-2007

124

## Gestion du temps Exemple des alarmes OSEK / VDX - exercice 12

- ❑ On dispose d'une alarme associée à la tâche T1 et au compteur de ticks d'horloge, avec une action par activation
- ❑ Donner le code pour activer T1 périodiquement 100 fois de suite (avec une périodicité de 50 ticks), avec un premier déclenchement au tick 10, et ce en utilisant :
  - (a) un déclenchement cyclique / absolu de l'alarme
  - (b) un déclenchement cyclique / relatif de l'alarme
  - (c) un déclenchement unique / absolu de l'alarme
  - (d) un déclenchement unique / relatif de l'alarme
  - Exhiber les cas pouvant poser des problèmes de régularité de l'activation
- ❑ Reprendre l'exercice (cas a) seulement) avec une alarme émettant un événement à la tâche
- ❑ Solution

Option STR – 2006-2007

125

## Gestion du temps Exemples de primitives

Primitive	xeC_B6	OSEK / VDX	VxWorks
Exécution / événement non périodique	Aucune	SetRelAlarm(aid,increment,0); SetAbsAlarm(aid,start,0)	taskDelay(ticks); timer_create(CLOCK_REALTIME,event_jandler,&tid); timer_settime(tid,flags, /* ab or rel */time, prevtime);
Exécution / événement périodique	startDelay(Numdelay,FirstDelay,Period,adFn);	SetRelAlarm(aid,incr,cycle); SetAbsAlarm(aid,start,cycle);	
Annulation / destruction	StopDelay(Numdelay,FirstDelay,Period,adFn);	CancelAlarm(aid);	timer_delete(tid);
Chiens de garde	WaitEvents(Mask,Timeout,&MaskOccured); P(NumSema,Timeout); PWithPrio(NumSema,Timeout); Receive(NumMbox,&message,Timeout);	Aucun	wid = wdCreate(); wdStart(wid,ticks,fnptr,param); wdCancel(wid); wdDelete(); semTake(semid,timeout); msgQReceive(mid,buffer,maxNBytes,timeout);
Consultation		GetAlarm(aid,&tick);	timer_gettime (tid,&time);

Option STR – 2006-2007

126

## Gestion des interruptions

- ❑ Définitions
  - Sollicitation externe → requête d'interruption
  - Exécution d'une routine de traitement d'interruption (ISR = Interrupt Service Routine)
  - Possibilité de priorités entre interruptions selon les architectures
  - En général, appel à l'ordonnanceur au retour de l'ISR
- ❑ Un ISR n'a pas le statut d'une tâche
  - Pas de contexte reconnu de l'exécutif (s'exécute soit dans le contexte de la tâche interrompue, soit dans un contexte spécifique)
- ❑ Restrictions sur les ISR
  - Ne doit **pas appeler des routines bloquantes** (n'est pas une tâche)
  - Doit être **courte** : mode non préemptif en général
    - Signaler un événement pour lancer une tâche, sauf si ISR très courte (ex: mise à jour horloge système)

Option STR – 2006-2007

127

## Gestion des interruptions

- ❑ Modes de gestion des périphériques
  - Sous interruption
    - pas/peu de latence dans le traitement des entrées sorties
    - peu de maîtrise des instants de demandes d'interruptions → difficulté à intégrer dans les méthodes d'analyse d'ordonnancement
  - Polling : examen périodique des registres de coupleurs de périphériques
    - plus de latence dans le traitement des entrées / sorties
    - plus facile à intégrer dans les méthodes d'analyse d'ordonnancement

Option STR – 2006-2007

128

## Gestion des interruptions Primitives typiques

- ❑ Connexion d'un ISR à un niveau d'interruption
- ❑ Masquage / démasquage des interruptions
  - Utilisable pour mettre en œuvre des sections critiques de courte durée
  - Avec mémorisation de l'état précédent (imbricable, gestion d'une pile) ou non

Option STR – 2006-2007

129

## Gestion des interruptions Interruptibilité du noyau

- ❑ Noyau non préemptible (masquage IT pour données partagées)
  - Appels systèmes par construction indivisibles
  - Problème de réactivité du noyau
    - Prise en compte des interruptions différée (**gigue**), voire perte d'interruptions
    - Latence max prise en compte interruption peut être longue (plus long appel système)
  - **Mauvaise propriété**
- ❑ Noyau préemptible
  - Noyau réactif, latence de prise en compte des interruptions plus faible
  - Indivisibilité des appels au noyau + difficile à assurer : on peut masquer/interdire les interruptions sur des **portions courtes** (cf ex. evt)
  - **Bonne propriété** = sections non préemptibles bornées et de courte durée

Option STR – 2006-2007

130

## Gestion des interruptions Exemples de primitives

Primitive	xec_86	OSEK / VDX	VxWorks
Interdiction		DisableAllInterrupts(); /* Pas de nesting */ SuspendAllInterrupts(); /* Nesting */	intLock();
Autorisation		EnableAllInterrupts(); /* Pas de nesting */ ResumeAllInterrupts(); /* Nesting */	intUnlock();
Connexion handler	setvect(); /* DOS */	Non spécifié	intConnect(vector,routine,param);

Option STR – 2006-2007

131

## Gestion de la mémoire

- ❑ Types de gestion de mémoire
  - Statique vs dynamique
  - Protégé vs non protégé
  - Quid de la pagination en contexte temps-réel ?

Option STR – 2006-2007

132

## Gestion de la mémoire Statique vs dynamique

- ❑ Gestion **statique** de mémoire
  - Les **nombre**s de tous les objets du noyau (tâches, sémaphores, boîtes à lettres) sont connus (ou bornés) au chargement du noyau
  - Les **tailles** de tous les objets du noyau (piles, tampons de réception des messages, zones de code, de données) sont connus (ou bornés)
  - Pas de routine de création d'objet / d'allocation de mémoire
  - Avantages
    - pas de pénurie de mémoire à l'exécution (sûreté)
    - temps d'accès aux objets constant et faible (tableaux)
  - Inconvénients
    - Peu flexible

Option STR – 2006-2007

133

## Gestion de la mémoire Statique vs dynamique

- ❑ Gestion **dynamique** de la mémoire
  - On ne connaît pas les nombres et/ou taille des objets
  - Routine de créations d'objets (tâches, sémaphores, ...) et ou d'allocation dynamique de mémoire
  - Avantages
    - flexibilité
  - Inconvénients (voir transparent suivant)
    - Temps d'accès aux objets plus élevé et plus difficile à prédire (listes)
    - Risque de pénurie de mémoire en cours d'exécution
    - Risque d'émiettement de la mémoire (fragmentation)
    - Durée de l'allocation de mémoire difficile à borner

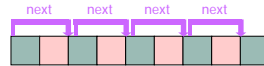
Option STR – 2006-2007

134

## Gestion de la mémoire Statique vs dynamique

- ❑ Gestion dynamique de mémoire (suite): ex, allocateur « first-fit »
  - Repérer les blocs occupés (pleins) des blocs libres (trous)
  - Chainage de blocs libres dans les trous eux-mêmes
  - First-fit : on retourne le premier bloc libre de la liste de taille suffisante

```
for (b = h->first; b != tail ; b = b->next) {
  if (b->size >= size) {
    break; /* block found */
  }
}
```



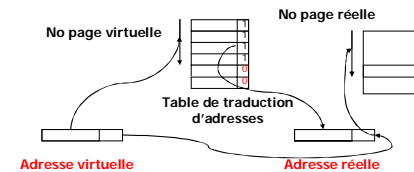
- Trous générés au fil des allocations/déallocations successives
- Temps d'allocation d'un bloc dépend de l'historique des allocations et de la fragmentation (au pire, parcours de toute la mémoire)
  - ➔ **Inadapté au temps-réel**, sauf si échéances élevées ou précautions pour éviter la fragmentation (allocation au démarrage)

Option STR – 2006-2007

135

## Gestion de la mémoire Protégé vs non protégé

- ❑ Gestion **protégée** de la mémoire
  - Tâches dans des espaces mémoire séparés (espaces d'adressage)
  - **Support matériel** (MMU, mécanisme de segmentation du processeur)
  - Exemple d'utilisation d'une MMU avec table des pages linéaires



- Dépassements de la zone adressable de la tâche détectée par le matériel, arrêt de la tâche (pas de modifications des autres tâches)

Option STR – 2006-2007

136

## Gestion de la mémoire Protégé vs non protégé

- ❑ Gestion **protégée** de la mémoire
  - Avantages
    - Résistance aux fautes de programmation
  - Inconvénients
    - Latence de traduction d'adresse (accès à la table des pages), non prévisibilité quand cache de traduction d'adresse - TLB)
    - Occupation mémoire supplémentaire pour les tables de traduction d'adresses
    - Latence de changement de contexte plus élevée (sauvegarde/restauration du contexte mémoire, rechargement TLB)
  - Remarque : on ne peut plus directement partager la mémoire entre tâches : besoin d'un support système (segments de mémoire partagée)

Option STR – 2006-2007

137

## Gestion de la mémoire Protégé vs non protégé

- ❑ Gestion **non protégée** de la mémoire
  - Tâches dans le même espace mémoire (accès direct à la mémoire réelle)
  - Avantages
    - temps d'accès à la mémoire constants
  - Inconvénients
    - Accès mémoire erronés non détectés, possibilités d'écrasement du code/données des autres tâches
    - ex: 

```
int tab[100];
for (i=0;i<100;i++) tab[2*i] = 0;
```

Option STR – 2006-2007

138

## Gestion de la mémoire Pagination à la demande ?

- ❑ Définition
  - Chargement en mémoire vive « à la demande » (swap in) quand une entrée est invalide dans la table de traduction (défaut de page)
  - Recopie en stockage secondaire (disque) quand la mémoire est pleine (swap out)
  - Intérêt : utilisation de tâches ayant de besoins en mémoire supérieurs aux capacités de la mémoire physique
- ❑ Quid de la pagination en contexte temps réel ?
  - temps d'accès à la mémoire difficile à prédire et potentiellement **très grand** (latence d'accès au disque) → **à proscrire en contexte temps-réel**
  - utilisé dans les systèmes d'exploitation **généralistes** (non dédiés temps-réel)
  - extensions de systèmes généralistes au temps-réel : primitives de **verrouillage des pages** en mémoire (on interdit leur transfert sur disque), ex : extensions temps-réel de POSIX

Option STR – 2006-2007

139

## Gestion de la mémoire Exemples de primitives

Primitive	xec_86	OSEK / VDX	VxWorks (non exhaustif)
<b>Allocation</b>	Aucune	Aucune	ad = memalign (alignment, nbytes); ad = malloc(nbytes); ad = valloc(nbytes); /* On page boundaries */ ad = calloc(nelem, elemsize); ad = realloc(adoldblock, newsize);
<b>Libération</b>	Aucune	Aucune	free(ad); cfree(ad);

Option STR – 2006-2007

140

---

---

## Quelques exécutifs temps-réel et/ou embarqués

---

---

---

---

## Plan

---

---

- ❑ Standardisation : extensions temps-réel de POSIX
  - Autre standardisation : OSEK/VDX (automobile) : les premiers produits sortent (e.g. OSEKWorks de WindRiver)
  - µitron : standard de-facto utilisé dans l'embarqué au Japon
- ❑ Le marché des exécutifs temps-réel (1999)
- ❑ Quelques exécutifs temps-réel industriels (liste non exhaustive ...)
  - VxWorks
  - pSOS
  - VRTX
  - QNX
  - Quelques autres exécutifs
- ❑ Bilan des exécutifs existants (1999)
- ❑ Critères de sélection d'un exécutif temps-réel

---

---

Option STR – 2006-2007

142

---

---

## Extensions temps-réel de POSIX

---

---

- ❑ POSIX
  - Standardisation de l'interface d'appel des systèmes UNIX
  - Standard IEEE 1003.1, datant de 1987, dans laquelle sont normalisées les interfaces de gestion :
    - Des processus
    - Des fichiers
    - Des entrées /sorties console
    - De l'environnement des processus
- ❑ Différentes extensions de POSIX au temps-réel
  - 1003.4 : extensions **temps-réel** à POSIX 1003.1 (aussi nommé 1003.1b)
  - 1003.4a : extensions au **multithread** de POSIX 1003.1

---

---

Option STR – 2006-2007

143

---

---

## Extensions temps-réel de POSIX : 1003.1b Ordonnancement

---

---

- ❑ Ordonnancement
  - Priorité par processus
  - Trois politiques d'ordonnancement, chacune associée à un intervalle de priorités (les intervalles peuvent ou non se recouvrir)
    - SCHED\_FIFO : ordonnancement par priorité avec un ordonnancement FIFO à priorité égale
    - SCHED\_RR : ordonnancement par priorité avec tranches de temps (round-robin) à priorités égales
    - SCHED\_OTHER : autre politique d'ordonnancement
  - Interface
    - sched\_setparam(pid,&param); /\* Param = priorité pour les 2 1ers ordos \*/
    - sched\_getparam(pid,&param);
    - sched\_setscheduler(pid,policy,&param);
    - sched\_getscheduler(pid);
    - sched\_yield();
    - sched\_get\_priority\_max/min(policy);

---

---

Option STR – 2006-2007

144

Extensions temps-réel de POSIX : 1003.1b  
Synchronisation

- ❑ Sémaphores (sémaphores à compteur)
  - `sem_init(&sem, shared, value);`
  - `sem_destroy(&sem);`
  - `sem_wait(&sem); sem_trywait(&sem);`
  - `sem_post(&sem);`
  - `sem_getvalue(&sem, &val);`
  - Gestion de files spécifiée pour SCHED\_FIFO et SCHED\_RR (par priorités)
- ❑ Signaux

Option STR – 2006-2007

145

Extensions temps-réel de POSIX : 1003.1b  
Communications

- ❑ Files de messages
  - taille bornée
  - flag pour paramétrer le caractère bloquant des émissions/réceptions
  - files de messages gérées par priorités, FIFO à priorités égales
  - `mq_send(mq_id, ptr, size, prio);`
  - `mq_receive(mq_id, ptr, len, &prio);`
  - `mq_notify(mq_id, notification);` /\* Pour être averti par signal arrivée d'un msg \*/

Option STR – 2006-2007

146

Extensions temps-réel de POSIX : 1003.1b  
Gestion mémoire (1/2)

- ❑ Verrouillage de mémoire
  - `mlockall(flags);` /\* Verrouille toutes les pages mappées \*/
    - Flag MCL\_CURRENT s'applique à toutes les pages mappées à l'instant t
    - MCL\_FUTURE s'applique au pages mappées dans le futur (positionner les deux flags si on veut TOUT verrouiller)
  - `munlockall();`
  - `mlock(addr, size);` /\* Verrouille un ensemble de pages \*/
  - `munlock(addr, size);`

Option STR – 2006-2007

147

Extensions temps-réel de POSIX : 1003.1b  
Gestion mémoire (2/2)

- ❑ Mapping de fichiers : permet l'accès direct d'un fichier à une zone d'adresses (pas de verrouillage)
  - `mmap(addr, len, prot, flags, filedesc, offset);`
  - `munmap(addr, len);`
  - `mprotect(addr, len, prot);` /\* Changement de droits d'accès \*/
  - `mmsync(addr, len, flags);` /\* Mise à jour fichier \*/
- ❑ Objets de mémoire partagée
  - `shm_open(name, flag, mode);` /\* Retourne un file descriptor, utilisé ensuite comme un fichier \*/
  - `shm_unlink(name);`

Option STR – 2006-2007

148

Extensions temps-réel de POSIX : 1003.1b  
Gestion du temps

- ❑ Temps
  - `clock_settime(clock_id,&time); clock_gettime(clock_id,&time);`
  - `clock_getres(clock_id,&res);`
- ❑ Timers (envoi signal quand déclenchement timer)
  - `timer_create(clock_id,&sigevent,&tidem_id); timer_delete(tid);`
  - `timer_settime(timer_id,flag (absolu/non),&value,&ovalue);` (value spécifie une périodicité)
  - `timer_gettime(timer_id,&value);`
  - `timer_getoverrun(timer_id);` /\* Dit si double signal \*/
  - `nanosleep(&delay,&reminder);` /\* Blocage en attente de signal \*/
- ❑ Notes
  - Standardise l'interface, pas l'implantation
  - Interruptibilité du noyau ?

Option STR – 2006-2007

149

Situation de l'offre industrielle (1999)

- ❑ Source : « systèmes d'exploitation temps-réel, exemples d'exécutifs industriels ». Techniques de l'ingénieur, R 8 052, 1999
- ❑ Offre industrielle importante, pas de réel leader
  - Systèmes propriétaires : 31%
  - VxWorks : 11%
  - pSOS : 9%
  - VRTX, OS9 : 8%
  - QNX, iRMX : 5%
  - Lynx-OS : 4%
  - Autres (multiples produits  $\leq 1\%$ ) : 19%
- ❑ Chiffres à prendre avec précautions (résultats d'enquête)
- ❑ Comparaisons difficiles à établir (difficulté d'obtention de documentations techniques)

Option STR – 2006-2007

150

Exemples d'exécutifs industriels  
VxWorks

- ❑ par WindRiver Systems
- ❑ Construit de manière modulaire autour du noyau wind
- ❑ Traitement des architectures multiprocesseurs (VxMP), des systèmes avec protection mémoire (VxMI), des architectures réseau (VindNet)
- ❑ Nombre de processeurs cible très important
- ❑ Interface wind + interface POSIX 1003.1b complète (API très riche)
- ❑ Outils associés
  - Tornado : environnement de développement sur machine hôte
  - Compilateur et débogueur
  - Simulateur (VxSim)
  - Observateurs très élaborés du comportement de la cible (WindView, StethoScope)
- ❑ Autre noyau du même vendeur : OSEKworks

Option STR – 2006-2007

151

Exemples d'exécutifs industriels  
VxWorks

- ❑ Gestion des tâches
  - Gestion et nommage dynamique des tâches
  - Signalisation asynchrone pour le traitement des erreurs
  - Ordonnancement préemptif par priorités (256 niveaux) et partage de temps sur un même niveau, possibilité de changer la priorité des tâches
  - Possibilité d'inhiber l'ordonnancement
- ❑ Synchronisation
  - Sémaphores d'exclusion mutuelle (mutex), sémaphores binaires, sémaphores à compteur
  - Mutex : sémaphores sans comptes avec les particularités suivantes :
    - héritage de priorités (PIP)
    - gestion des accès aux ressources emboîtés
    - protection contre la suppression des tâches en section critique
  - Sémaphores binaires : mutex sans ces 3 particularités

Option STR – 2006-2007

152

## Exemples d'exécutifs industriels VxWorks

- ❑ Communication
  - Boîtes à lettres avec messages de taille variable
  - Tubes (pipes)
  - Sockets (TCP/IP) et appels de procédure à distance sont de base dans VxWorks
- ❑ Gestion du temps
  - Chiens de garde auquel on peut connecter une fonction qui s'exécutera à l'expiration (mécanisme pour exécution de tâches périodiques)
  - Chiens de garde dans les services bloquants (sémaphores, mutex, messages)
- ❑ Gestion de mémoire
  - Gestion dynamique, support MMU optionnel (bibliothèque)
- ❑ Gestion des architectures multiprocesseurs

Option STR – 2006-2007

153

## Exemples d'exécutifs industriels pSOSystem 3

- ❑ par WindRiver systems (anciennement par Integrated Systems Inc.)
- ❑ Composé du noyau pSOS+ (petit noyau) et de nombreux composants logiciels (architecture configurable)
- ❑ Nombreuses architectures cibles (hôtes = Unix-Solaris/Windows NT)
- ❑ Librairie Posix (sous-ensemble)
- ❑ Environnement de développement pRISM+, outil d'instrumentation pMONT+, outil de mise au point pROBE+, système de gestion de fichiers pHILE+

Option STR – 2006-2007

154

## Exemples d'exécutifs industriels pSOSystem 3

- ❑ Gestion des tâches
  - Gestion et nommage dynamique des tâches
  - Ordonnancement préemptif à priorités (255 priorités) avec round-robin pour les tâches de même priorité
  - Gestion des tâches périodiques avec des réveils périodiques
- ❑ Synchronisation
  - Signalisation synchrone par événements (16 par tâche) ; attente de plusieurs événements possibles (ET ou OU)
  - Possibilité de signalisation asynchrone vers une tâche (ASR)
  - Exclusion mutuelle par sémaphores à compteurs. Pas de prise en compte du phénomène d'inversion de priorités
- ❑ Communications
  - Boîtes à lettres de taille fixe (16 octets) et variable. Attente des messages en FIFO ou par priorités

Option STR – 2006-2007

155

## Exemples d'exécutifs industriels pSOSystem 3

- ❑ Gestion du temps
  - Signalement d'événements après un intervalle de temps, périodiquement ou à une date donnée
  - On peut demander le réveil d'une tâche après un intervalle ou à une date donnée
  - Chiens de garde pour les services bloquants
- ❑ Gestion de la mémoire
  - Dynamique
  - Support MMU selon les architectures cibles
- ❑ Support pour les architectures multiprocesseurs
- ❑ Support réseau
  - TCP/IP (pNA+), NFS, RPC (pRPC+)

Option STR – 2006-2007

156

## Exemples d'exécutifs industriels VRTX

- ❑ par Mentor Graphics
- ❑ Deux solutions distinctes
  - VRTXsa (scalable architecture) : système d'exploitation modulaire configurable pour les systèmes complexes.
  - VRTXmc (micro-controller) : noyau temps-réel compact conçu pour les applications à forte contrainte mémoire (ROM et RAM). Pour les systèmes simples
  - Compatibilité ascendante de VRTXmc avec VRTXsa pour réutilisation de code
- ❑ Outils associés :
  - Environnement de développement (Spectra) comprenant un débogueur (XRAY), un simulateur (Virtual Target) et un analyseur de performances (Xpert Profiler)

Option STR – 2006-2007

157

## Exemples d'exécutifs industriels VRTXsa

- ❑ Nananoyau VRTX
- ❑ Pile de protocole TCP/IP, applications et services associés (DHCP, BOOTP, PPP, NFS, telnet, ftp, ...)
- ❑ Gestion de fichiers
- ❑ Gestion d'entrées-sorties
- ❑ Modularité
  - TCP/IP, gestion de fichiers, gestion d'entrées sorties peuvent ne pas être générés
  - Possibilité de ne pas générer un sous-ensemble des appels système (e.g. queues ou mutex)
- ❑ Noyau
  - Ordo à priorités (255), round-robin à prio. égale, événements (32), mutex avec PIP, boîtes à lettres de taille fixe ou variable, gestion mémoire statique et dynamique, support MMU optionnel

Option STR – 2006-2007

158

## Exemples d'exécutifs industriels QNX Neutrino

- ❑ par QNX
- ❑ Noyau Neutrino modulaire pour systèmes embarqués complexes
- ❑ Architecture micro-noyau
  - micro-noyau : services essentiels (ordonnancement, IPC, synchro.). Communique avec les modules du système d'exploitation par messages
  - modules pour les services non essentiels
- ❑ Services pour la **tolérance aux fautes** : protection mémoire + « high availability manager » : messages de vie (heartbeats), checkpointing/restart automatique, analyse post-mortem
- ❑ Interface conforme aux extensions temps-réel de POSIX
- ❑ Services
  - gestion graphique, Java, gestion de fichiers (QNX, Linux, DOS, NFS, ...), réseau (piles TCP/IP « tiny » et complète), support pour différents périphériques (USB, audio, PCI, disques IDE, SCSI, ...)

Option STR – 2006-2007

159

## Exemples d'exécutifs industriels Quelques autres exécutifs (1/2)

- ❑ Lynx-OS de LynuxWorks
  - Entièrement conforme au standard POSIX (1003.1, .1b extensions temps-réel .1c extensions aux threads)
  - Compatibilité binaire Linux
- ❑ Nucléus (Accelerated Technologies - Embedded systems division de Mentor Graphics)
  - Code source disponible, pas de royalties
  - Noyaux : Nucléus PLUS, Nucléus OSEK, Nucléus µiPLUS
- ❑ eCOS : Embedded Configurable Operating System (Red Hat, Inc.)
  - Noyau open-source

Option STR – 2006-2007

160

## Exemples d'exécutifs industriels Quelques autres exécutifs (2/2)

- ❑ RTEMS
  - Noyau open-source multi-processeur
  - Support et services par OAR (On-line Application Research)
  - Interfaces POSIX, µitron, RTEMS
- ❑ Jaluna (Jaluna SA, anciennement ChorusOS de Sun micro-systems)
  - Noyau open-source depuis fin octobre 2002
  - Architecture à base de micro-noyau : scheduler modulaire, gestion de mémoire virtuelle, communication et synchronisation, système de conception de drivers de périphériques portable
  - Surcouche RT-POSIX
  - Support pour la haute disponibilité

Option STR – 2006-2007

161

## Critères de sélection d'un exécutif temps-réel/embarqué (1/3)

- ❑ A t'on besoin d'un exécutif ?
  - + Développement d'applications plus rapide (gain en **productivité**). On n'a pas à gérer le partage de processeur « à la main »
  - Place **mémoire** supplémentaire, surcoût en **temps** d'exécution
- ❑ Critères de sélection d'un exécutif
  1. Cibles supportées
  2. Langages de programmation supportés (les plus courants : C,C++,Ada)
  3. Taille mémoire en ROM et RAM (footprint)
    - Noyaux modulaires → taille du noyau variable
    - Problème : identifier le contenu correspondant aux tailles minimales annoncées par le constructeur

Option STR – 2006-2007

162

## Critères de sélection d'un exécutif temps-réel/embarqué (2/3)

4. Services fournis par l'exécutif (noyau)
  - Ordonnanceur : adapté au temps-réel
  - Services de synchronisation : quelle gestion de file ? (FIFO, priorités), mécanismes pour l'inversion de priorité ?
  - Gestion mémoire : Gestion MMU ? Gestion mémoire statique ou dynamique ?
  - Gestion du temps : lancement de tâches périodiques
  - Interruptibilité du noyau
  - Quels services est-on obligés d'inclure dans le noyau. Tous ?
5. Composants logiciels fournis en plus du noyau
  - Piles de protocoles, bases de données temps-réel, services Web disponibles ?
  - Temps de portage si ces services ne sont pas disponibles

Option STR – 2006-2007

163

## Critères de sélection d'un exécutif temps-réel/embarqué (3/3)

6. Performances : existence de benchmarks, mais :
  - Signification des chiffres : quelle plate-forme, quelles conditions de mesure ? chiffres au-mieux, au-pire, moyen ? Tests en boucle (tout en cache !), présence d'interruptions, état noyau (nb obj, fragmentation)
  - Latence et fréquence des interruptions
7. Existence de drivers de périphériques
8. Existence d'une chaîne de développement
9. Support technique
10. Services assurés par la société (portage sur nouvelles architectures)
11. Nature du produit (code source ou binaire)
12. Licence/prix (coût par système vendu ?)
13. Réputation (nb de licences vendues, clients)
14. Contraintes « métier » : certification

Option STR – 2006-2007

164

---

## Vérification des contraintes de temps (analyse d'ordonnançabilité)

---

---

## Vérification des contraintes temporelles Introduction (1/2)

---

- Rôle
  - Formules mathématiques ou algorithmes permettant de vérifier (prouver) que les tâches respecteront leurs contraintes de temps (ex: échéances)
- Classification
  - Vérification hors-ligne (avant exécution)
    - critères analytiques calculables (CN, CS, CNS)
    - simulation
    - cibles = systèmes temps-réel strict
  - Vérification en-ligne (pendant exécution)
    - tests d'acceptation en-ligne pour savoir si on accepte de nouvelles tâches
    - risque de rejet de tâches → cibles = systèmes temps-réel souple

---

Option STR – 2006-2007

166

---

## Vérification des contraintes temporelles Introduction (2/2)

---

- Données d'entrée des méthodes de vérification : modèle du système
  - Connaissance des informations sur les tâches (modèle de tâches)
    - arrivée des tâches: périodique, sporadique, apériodique (dates d'arrivées)
    - synchronisations : précédences, exclusions entre tâches
    - temps d'exécution au pire-cas (WCET)
  - Architecture (mono-processeur ou multi processeur)
  - Ces données sont connues statiquement pour les analyses d'ordonnançabilité hors-ligne
- Sorties
  - Verdict sur le respect des contraintes de temps pour un algorithme d'ordonnancement donné
  - Données pour l'ordonnancement (priorités, plan)

---

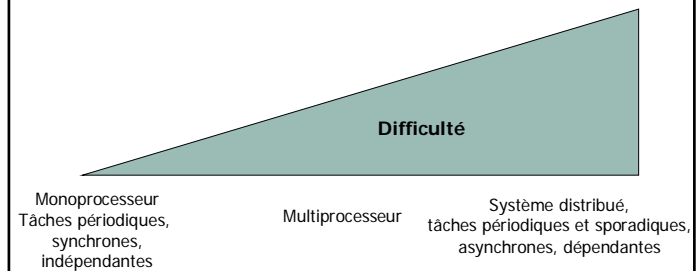
Option STR – 2006-2007

167

---

## Vérification des contraintes temporelles Complexité de la vérification (1/2)

---



De manière générale, le problème d'ordonnançabilité est NP-difficile

---

Option STR – 2006-2007

168

## Vérification des contraintes temporelles Complexité de la vérification (2/2)

Monoprocasseur			Multiprocasseur		
<b>Indep.</b>	NoPr   Lmax Pr   Lmax NoPr, Ri   Lmax	Poly Poly NP-hard	<b>NoPr, 2 proc</b>	NoPr, prec, Ci=1   Lmax NoPr, prec, ress NoPr, Ci <= 1   Lmax NoPr, Ci=1, 1 ress	Poly NP-hard NP-hard NP-hard
<b>Préc.</b>	NoPr, prec   Lmax NoPr, prec, Ri   Lmax Pr, prec, Ri   Lmax	Poly NP-hard Poly	<b>NoPr, N Proc</b>	NoPr, prec, Ci=1	NP-hard
<b>Ress.</b>	Per + sémas Pr, Ri, Ci=1   Lmax NoPr, prec, Ri, Ci=1   Lmax	NP-hard Poly Poly	<b>Pr, N proc</b>	Pr   mmiss	NP-hard

Option STR – 2006-2007

169

## Vérification de contraintes temporelles Plan (1/2)

- ❑ Faisabilité pour **tâches apériodiques** (à dates d'arrivée connues) et ordonnancements non préemptifs
- ❑ Faisabilité pour **tâches périodiques et ordos préemptifs à priorités**
  - Rappel des notations et hypothèses
  - Approches par simulation
  - Condition nécessaire d'ordonnançabilité
  - Vérification pour ordonnancements à priorités fixes
    - Condition pour Rate Monotonic (CS) et Deadline Monotonic (CS)
    - Condition générale pour algorithmes à priorités fixes : Response Time Analysis (CNS)
  - Vérification pour ordonnancements à priorités dynamiques
    - Condition pour EDF pour tâches à échéance sur requête (CNS)
    - Condition pour EDF pour tâches à échéance  $\leq$  période : Processor Demand Analysis (CNS)

Option STR – 2006-2007

170

## Vérification de contraintes temporelles Plan (2/2)

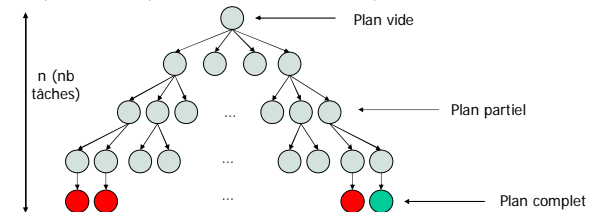
- ❑ Faisabilité pour tâches avec **partage de ressources**
  - Ordonnancements non préemptifs
  - Ordonnancements préemptifs à priorités statiques (RM, DM, RTA) et dynamique (EDF, Processor Demand)
- ❑ Prise en compte des coûts système
- ❑ Méthodes d'obtention des WCET (Worst-Case Execution Times) (Ci)

Option STR – 2006-2007

171

## Faisabilité pour tâches apériodiques en non préemptif

- ❑ Complexité du problème
  - Le problème consistant à trouver un ordonnancement non préemptif pour un ensemble de tâches apériodiques à dates d'arrivées arbitraires connues a priori est un problème **NP-complet**
- ❑ Le problème exprimé sous forme d'une exploration d'arbre



Option STR – 2006-2007

172

## Faisabilité pour tâches apériodiques en non préemptif

### □ Solutions

- Exploration **exhaustive** de toutes les séquences d'ordonnement possibles  $O(n!)$  : trop complexe dans le cas général
- Réduction de l'arbre de recherche
  - **Elagage** (pruning)
    - Exemple : algorithme de Bratley
  - **Heuristiques** pour guider le parcours de l'arbre

□ Complexité de la recherche → orienté vers la vérification hors-ligne

Option STR – 2006-2007

173

## Faisabilité pour tâches apériodiques en non préemptif Algorithme de Bratley (1971) (1/3)

### □ Modèle de tâches

- Tâches apériodiques avec date d'arrivée connues

### □ Principe de fonctionnement

- Construction d'un plan sans préemption

### □ Principe

- A un nœud de l'arbre est associé un plan provisoire
- Exploration à partir d'un nœud en considérant toutes les tâches encore possibles
- On élague une branche (pruning) dans deux cas :
  - on a trouvé un plan dans lequel les échéances sont respectées
  - l'ajout de tout nœud sur le chemin courant entraîne un dépassement d'échéance

Option STR – 2006-2007

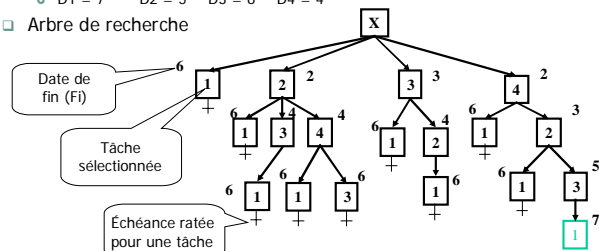
174

## Faisabilité pour tâches apériodiques en non préemptif Algorithme de Bratley (1971) (2/3)

### □ Exemple

- $A1 = 4$     $A2 = 1$     $A3 = 1$     $A4 = 0$
- $C1 = 2$     $C2 = 1$     $C3 = 2$     $C4 = 2$
- $D1 = 7$     $D2 = 5$     $D3 = 6$     $D4 = 4$

### □ Arbre de recherche



Option STR – 2006-2007

175

## Faisabilité pour tâches apériodiques en non préemptif Algorithme de Bratley (1971) (3/3)

### □ Propriétés de l'algorithme

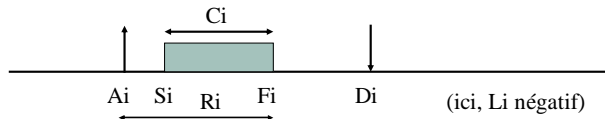
- **Non optimal** (s'arrête au premier ordonnancement faisable, pas nécessairement le meilleur)
- En moyenne, technique d'élagage très efficace, mais complexité au pire est toujours  $O(n!)$
- Utilisable pour de la vérification **hors-ligne** seulement

Option STR – 2006-2007

176

## Analyse d'ordonnancement pour tâches périodiques Notations (1/2)

- Arrivée Ai
- Temps de calcul maximum sans préemption Ci
- Échéance (deadline) : relative ou absolue Di
- Date de début (start time) et fin (finish time) Si, Fi
- **Temps de réponse** Ri = Fi - Ai
- Retard (lateness)  $Li = (Fi - Di)$
- Laxité (slack time)  $xi(t) = Di - (t + Ci - ci(t))$   
 $Xi = xi(Ai) = (Di - Ai - Ci)$



Option STR – 2006-2007

177

## Analyse d'ordonnancement pour tâches périodiques Notations (2/2)

- Arrivées des tâches
  - **Périodiques** : arrivée à intervalles réguliers ( $Pi$ )
    - Date d'activation initiale, **offset**  $Oi$
    - Si pour tout  $i, j$   $Oi = Oj$ , tâches synchrones
    - Si  $Di = Pi$ , tâche à échéance sur requête
- Instant critique : date pour laquelle une arrivée de tâches produira son pire temps de réponse

Option STR – 2006-2007

178

## Analyse d'ordonnancement pour tâches périodiques Hypothèses

- Hypothèses explicites
  1. Tâches périodiques de période  $Pi$
  2. Temps d'exécution pire-cas  $Ci$  constant pour toutes les instances
  3. Échéance sur requête ( $Di = Pi$ )
  4. Tâches indépendantes (pas de synchronisation/communication)
  5. Tâches synchrones ( $Oi = 0$ )
- Hypothèses implicites
  5. Pas de suspensions (volontaires ou involontaires) des tâches
  6. Une tâche peut être lancée dès son arrivée ( $Ai$ ) : pas de gigue au démarrage
  7. Surcoûts du système d'exploitation (démarrage tâche, préemption) nuls
- Hypothèses considérées pendant **toute la partie sur l'ordonnancement de tâches périodiques**, sauf mention contraire explicite.

Option STR – 2006-2007

179

## Analyse d'ordonnancement pour tâches périodiques Vérification par simulation

- Périodicité des tâches → ordonnancement cyclique (pas la peine de simuler à l'infini)
- Durée de la période d'étude (ou pseudo-période)
  - Tâches synchrones :  $[0, PPCM(Pi)]$
  - Tâches échelonnées (au moins un offset  $Oi$  non nul)
    - $[\min(Oi), \max(Oi, Oj + Dj)] + 2 * PPCM(Pi)$
- Evaluation
  - Valide quel que soit l'algorithme d'ordonnancement
  - Période d'étude peut être très longue selon les relations entre périodes
  - Adapté aux ordonnancements hors-ligne

Option STR – 2006-2007

180

Analyse d'ordonnancement pour tâches périodiques  
Condition nécessaire d'ordonnancement

- Notion de charge processeur

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

- Condition nécessaire d'ordonnancement :  $U \leq 1$ 
  - Cette condition n'est pas une condition suffisante (pas pour tous les ordonnancements)

Analyse d'ordonnancement pour tâches périodiques  
Faisabilité pour Rate Monotonic (1/2)

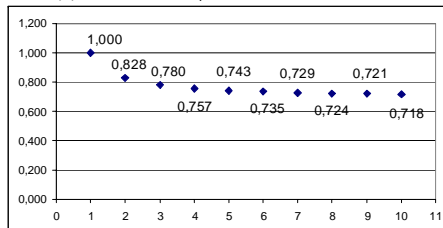
- Définition RM : la priorité d'une tâche est inversement proportionnelle à sa période (conflits résolus arbitrairement)
- Propriété RM : optimal parmi les algorithmes à priorités fixes pour des tâches périodiques indépendantes à échéance sur requête ( $D_i = P_i$ ).

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

- faible complexité  $O(n)$
- condition **suffisante** seulement (si  $U \in [U_{ub}, 1]$ , le jeu de tâches **peut** respecter ses échéances)
- s'applique aussi quand les tâches ne sont pas synchrones

Analyse d'ordonnancement pour tâches périodiques  
Faisabilité pour Rate Monotonic (2/2)

- Variation du seuil de charge en fonction de n (limite =  $\ln(2) = 0,69314718$ )



- Cas particulier : périodes des tâches sont des **harmoniques** (pour tout  $i, j$  tq  $P_i < P_j$ ,  $P_j = k P_i$  avec k entier) :  $U_{ub} = 1$

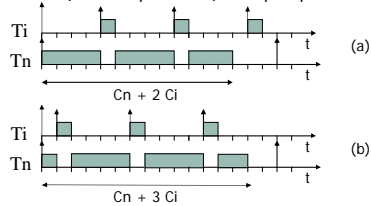
Analyse d'ordonnancement pour tâches périodiques  
Rate Monotonic : preuve d'optimalité (1/6)

- Démarche on montre que :

- Instant critique pour une tâche  $T_i$  = quand la date d'arrivée de  $T_i$  est simultanée avec les tâches de priorités supérieures
- En se plaçant à l'instant critique, si un jeu de tâche est faisable avec un assignement arbitraire de priorités, il l'est également avec RM
  - Démonstration pour 2 tâches
  - Extension à n tâches (pas fait ici)

Analyse d'ordonnancement pour tâches périodiques  
Rate Monotonic : preuve d'optimalité (2/6)

1. Instant critique pour une tâche  $T_i$  = quand la date d'arrivée de  $T_i$  est simultanée avec les tâches de priorités supérieures
  - o Soit la tâche  $T_n$  (la moins prioritaire) et  $T_i$  plus prioritaire que  $T_n$



- o Si on avance la date  $A_i$ , la date de fin de  $T_n$  peut augmenter. La date de fin la plus grande est quand  $A_i = A_n$
- o Transposable à toutes les tâches  $T_1, \dots, T_n$ , d'où le résultat

Option STR – 2006-2007

185

Analyse d'ordonnancement pour tâches périodiques  
Rate Monotonic : preuve d'optimalité (3/6)

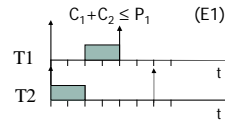
1. Si jeu de tâches faisable avec des priorités arbitraires, l'est sous RM (démonstration pour  $n=2$ )
  - o Deux tâches  $T_1$  et  $T_2$  avec  $P_2 > P_1$
  - o Démarche
    - o Assignment des priorités selon des priorités arbitraires non RM (un seul choix pour 2 tâches !)
    - o Etablissement de la condition de faisabilité dans ce cas (E1)
    - o Assignment des priorités selon RM
    - o On montre que si E1 vérifié, alors le système est faisable également avec l'assignment de priorités RM

Option STR – 2006-2007

186

Analyse d'ordonnancement pour tâches périodiques  
Rate Monotonic : preuve d'optimalité (4/6)

- Assignment des priorités non-RM :  $\text{prio}(T_2) > \text{prio}(T_1)$ 
  - o On se place à l'instant critique
  - o Le système est faisable si



- o On montre que si E1 est vérifié, alors le système est aussi faisable en assignant les priorités selon RM

- Assignment des priorités selon RM :  $\text{prio}(T_1) > \text{prio}(T_2)$ 
  - o Soit  $F$  le nombre de périodes de  $T_1$  intégralement contenues dans  $P_2$

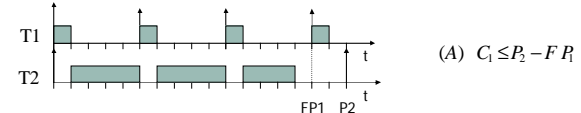
$$F = \left\lfloor \frac{P_2}{P_1} \right\rfloor \quad (F \geq 1)$$

Option STR – 2006-2007

187

Analyse d'ordonnancement pour tâches périodiques  
Rate Monotonic : preuve d'optimalité (5/6)

- A.  $C_1$  suffisamment court pour que toutes les instances de  $T_1$  arrivant dans l'intervalle  $[0, F P_1]$  soient terminées avant la date  $P_2$



- o Système est ordonnancement si la condition (E2) est vérifiée (cf figure)  $(F+1)C_1 + C_2 \leq P_2$  (E2)

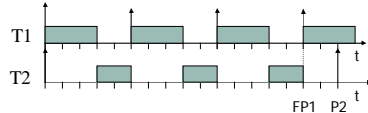
- o On montre que si (E1) vérifié (non-RM), alors (E2) vérifié (RM)
  - o D'après (E1) :  $C_1 + C_2 \leq P_1$ , soit en multipliant par  $F$  :  $FC_1 + FC_2 \leq FP_1$
  - o Comme  $F \geq 1$ ,  $FC_1 + C_2 \leq FC_1 + FC_2 \leq FP_1$
  - o En ajoutant  $C_1$  des deux cotés  $(F+1)C_1 + C_2 \leq FP_1 + C_1$
  - o Comme cas (A),  $C_1 \leq P_2 - FP_1$ , d'où  $(F+1)C_1 + C_2 \leq FP_1 + P_2 - FP_1$ , soit  $(F+1)C_1 + C_2 \leq P_2$ , ce qui vérifie (E2)

Option STR – 2006-2007

188

### Analyse d'ordonnancement pour tâches périodiques Rate Monotonic : preuve d'optimalité (6/6)

- B. C1 n'est pas suffisamment court pour que toutes les instances de T1 arrivant dans l'intervalle [0, F P1] soient terminées avant la date P2



$$(B) C_1 \geq P_2 - F P_1$$

- Système est ordonnable si la condition (E3) est vérifiée (cf figure)
 
$$F C_1 + C_2 \leq F P_1 \quad (E3)$$
- On montre que si (E1) vérifié (non-RM), alors (E3) vérifié (RM)
  - D'après (E1) :  $C_1 + C_2 \leq P_1$ , soit en multipliant par F :  $FC_1 + FC_2 \leq FP_1$
  - Comme  $F \geq 1$ ,  $FC_1 + C_2 \leq FC_1 + FC_2 \leq FP_1$ , d'où (E3) est vérifiée
- On a montré que dans les cas A et B, si système faisable avec des priorités arbitraires, l'est quand priorités assignées par RM
- Généralisable au cas de n tâches

Option STR – 2006-2007

189

### Analyse d'ordonnancement pour tâches périodiques Faisabilité pour Deadline Monotonic

- Hypothèse : hypothèses précédentes sauf H3 :  $D_i \leq P_i$
- Définition DM [Leung & Whitehead, 1985]
  - La priorité d'une tâche est inversement proportionnelle à son échéance relative (conflits résolus arbitrairement)
- Propriété DM
  - Optimal dans la classe des algorithmes à priorités fixes pour des tâches périodiques indépendantes à échéance  $\leq$  période
- Condition de faisabilité

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

- faible complexité  $O(n)$
- condition **suffisante** seulement (si  $\sum (C_i/D_i) > U_{ub}$ , le jeu de tâches **peut** respecter ses échéances)
- s'applique aussi quand les tâches ne sont pas synchrones

Option STR – 2006-2007

190

### Analyse d'ordonnancement pour tâches périodiques Response Time Analysis (1/4)

- Response time analysis (RTA, analyse de temps de réponse) [Joseph & Pandya, 1986, Audsley & al, 1993]
  - Applicable pour **tous** les ordonnancements à priorité statique, quel que soit l'assignation des priorités (RM, DM, autre)
  - Condition **nécessaire et suffisante**
  - Utilisable pour tout  $D_i$  (même si non inférieur à  $P_i$  : dans ce cas, RM et DM ne sont plus optimaux)
  - S'applique aussi quand les tâches ne sont pas synchrones, mais est alors une condition suffisante seulement
- Principe
  - Calcul du temps de réponse  $R_i$  de chaque tâche à son instant critique (calcul itératif)
  - Vérification pour tout  $i$  que  $R_i \leq D_i$

Option STR – 2006-2007

191

### Analyse d'ordonnancement pour tâches périodiques Response Time Analysis (2/4)

- Introduction au calcul de temps de réponse  $R_i$ 

$R_i = C_i$  (temps d'exécution pire-cas de  $T_i$ )  
+  $I_i$  (**interférences** dues à des préemptions par tâches plus prioritaires)

Nombre maximal de préemptions subies par une instance de  $T_i$  :

$$\sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil$$

Interférence correspondante ( $C_j$  pour chaque préemption par  $T_j$ )

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil C_j$$

Temps de réponse  $R_i$

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil C_j$$
- Pas de solution simple ( $R_i$  apparaît des deux cotés)
  - Solution  $R_i =$  plus petite valeur telle que l'équation est vérifiée
  - Pas besoin de vérifier l'équation  $\forall R_i \leq D_i$  (seulement quand le nombre de préemptions augmente)

Option STR – 2006-2007

192

Analyse d'ordonnancement pour tâches périodiques  
Response Time Analysis (3/4)

- Algorithme de calcul des Ri (itératif)

$$w_i^0 = C_i$$

$$w_i^{k+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_j^k}{P_j} \right\rceil C_j$$

- Quand la série converge, on a calculé le temps de réponse Ri
- Le système est ordonnancement quand  $R_i \leq D_i$

Analyse d'ordonnancement pour tâches périodiques  
Response Time Analysis (4/4)

- Exemple

	Ci	Pi	prio
T4	3	10	
T3	1	5	
T2	2	20	
T1	2	20	

$$w_i^0 = C_i = 2$$

$$w_1^1 = C_1 + \left\lceil \frac{C_4}{P_4} \right\rceil C_4 + \left\lceil \frac{C_3}{P_3} \right\rceil C_3 + \left\lceil \frac{C_2}{P_2} \right\rceil C_2 = 2 + \left\lceil \frac{2}{10} \right\rceil 3 + \left\lceil \frac{2}{5} \right\rceil 1 + \left\lceil \frac{2}{20} \right\rceil 2 = 2 + 3 + 1 + 2 = 8$$

$$w_1^2 = C_1 + \left\lceil \frac{8}{P_4} \right\rceil C_4 + \left\lceil \frac{8}{P_3} \right\rceil C_3 + \left\lceil \frac{8}{P_2} \right\rceil C_2 = 2 + \left\lceil \frac{8}{10} \right\rceil 3 + \left\lceil \frac{8}{5} \right\rceil 1 + \left\lceil \frac{8}{20} \right\rceil 2 = 2 + 3 + 2 + 2 = 9$$

$$w_1^3 = C_1 + \left\lceil \frac{9}{P_4} \right\rceil C_4 + \left\lceil \frac{9}{P_3} \right\rceil C_3 + \left\lceil \frac{9}{P_2} \right\rceil C_2 = 2 + \left\lceil \frac{9}{10} \right\rceil 3 + \left\lceil \frac{9}{5} \right\rceil 1 + \left\lceil \frac{9}{20} \right\rceil 2 = 2 + 3 + 2 + 2 = 9$$

Analyse d'ordonnancement pour tâches périodiques  
Exercice 13

- Soit le jeu de tâches suivant. Est-ce qu'il est ordonnancement en utilisant un ordonnanceur à priorités fixes ?

	Ci	Pi	Di
T4	1	4	3
T3	1	5	4
T2	2	6	5
T1	1	11	10

[Solution](#)

Analyse d'ordonnancement pour tâches périodiques  
Faisabilité pour EDF ( $D_i = P_i$ )

- Définition EDF [Liu & Layland, 1973]
  - A un instant donné, la tâche la plus prioritaire (parmi les tâches prêtes) est celle dont l'échéance absolue est la plus proche
- Propriété EDF
  - Optimal dans la classe des algorithmes à priorités dynamiques pour des configurations de tâches périodiques indépendantes avec échéance  $\leq$  période
- Condition de faisabilité (valide pour  $D_i = P_i$ )

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

- faible complexité  $O(n)$
- condition nécessaire et suffisante
- s'applique aussi quand les tâches ne sont pas synchrones

Analyse d'ordonnabilité pour tâches périodiques  
Faisabilité pour EDF ( $D_i \leq P_i$ )

□ Conditions de faisabilité ( $D_i \leq P_i$ )

- condition **suffisante**  $\sum_{i=1}^n \frac{C_i}{D_i} \leq 1$

- condition **nécessaire**  $\sum_{i=1}^n \frac{C_i}{P_i} \leq 1$

Analyse d'ordonnabilité pour tâches périodiques  
Faisabilité pour EDF : processor demand (1/6)

□ Processor demand approach [Baruah et al, 90]

□ Principe

- Calcul de la **demande totale en processeur** sur un intervalle  $[0, L]$
- Vérification pour tout  $L$  que la demande en processeur est inférieure ou égale à  $L$

□ Plan des explications sur « Processor demand »

- Cas  $D_i = P_i$ 
  - Expression de la demande en processeur
  - Notion de « busy period »
  - Condition de faisabilité
- Extension au cas  $D_i \leq P_i$

Analyse d'ordonnabilité pour tâches périodiques  
Processor demand - cas  $D_i = P_i$  (2/6)

□ Demande en processeur

- Définition : demande en processeur sur  $[t, t+L]$  = quantité de calcul demandée par les  $T_i$  sur  $[t, t+L]$  **qui doivent se terminer avant  $t+L$**
- Demande en processeur sur l'intervalle  $[0, L]$  :

$$C_p(0, L) = \sum_{i=1}^n \left\lfloor \frac{L}{P_i} \right\rfloor C_i$$

- $\lfloor L/P_i \rfloor$  est le nombre de fois ou la tâche  $P_i$  arrive dans l'intervalle  $[0, L]$  **et doit se terminer dans l'intervalle** (d'où le  $\lfloor \cdot \rfloor$ , on ne compte pas les arrivées se produisant dans l'éventuel morceau de période final)
- $C_i$  est son temps d'exécution à chaque lancement

□ Le système est ordonnable si pour tout  $L$  positif [Jeffay & Stone]

$$C_p(0, L) = \sum_{i=1}^n \left\lfloor \frac{L}{P_i} \right\rfloor C_i \leq L \quad (e1)$$

Analyse d'ordonnabilité pour tâches périodiques  
Processor demand - cas  $D_i = P_i$  (3/6)

□ Remarques

- Ordo. cyclique : **inutile de tester l'équation (e1) pour les  $L > \text{PPCM}(P_i)$**
- $C_p(0, L)$  fonction par paliers: à tester uniquement aux arrivées de tâches

□ Notion de busy period

- Plus petit intervalle  $[0, L]$  dans lequel toutes les tâches arrivant dans l'intervalle  $[0, L]$  sont **entièrement** exécutées, c'est à dire

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

- $\lceil L/P_i \rceil$  est le nombre de fois ou la tâche  $P_i$  arrive dans l'intervalle  $[0, L]$
- $W(L)$  = charge de travail demandée dans l'intervalle  $[0, L]$
- Busy period  $B_p = \min\{L \mid W(L) = L\}$
- Busy period = période d'activité ininterrompue du processeur (peut être infinie si le système n'est pas ordonnable)
- Résultat : **inutile de tester l'équation (e1) pour les  $L > B_p$**

Analyse d'ordonnabilité pour tâches périodiques  
Processor demand - cas  $D_i = P_i$  (4/6)

- Calcul de la durée Bp de la busy period
  - Busy period  $Bp = \min\{L \mid W(L) = L\}$
  - Algorithme (itératif)
    - $L = \sum C_i$ ;  $L' = W(L)$ ;
    - $H = \text{PPCM}(P_1, \dots, P_n)$
    - while ( $L' \neq L$  &&  $L' \leq H$ ) {
    - $L = L'$ ;
    - $L' = W(L)$ ;
    - }
    - if ( $L' \leq H$ )  $Bp = L$ ; else  $Bp = \infty$ ;
- Note : quand le système n'est surchargé, fin de busy period =
  - soit début de période de non occupation du processeur (idle time)
  - soit arrivée d'une tâche périodique

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

Analyse d'ordonnabilité pour tâches périodiques  
Processor demand - cas  $D_i = P_i$  (5/6)

- Condition de faisabilité : condition **nécessaire et suffisante**
  - Pour tout L correspondant à une date d'arrivée de tâches dans l'intervalle  $\min(Bp, \text{hyperpériode})$

$$C_p(0, L) = \sum_{i=1}^n \left\lfloor \frac{L}{P_i} \right\rfloor C_i \leq L$$

- NB : n'est pas intéressant en tant que tel pour le cas  $D_i = P_i$  : on a déjà une condition nécessaire et suffisante, de complexité moindre (condition sur la charge)

Analyse d'ordonnabilité pour tâches périodiques  
Processor demand - extension au cas  $D_i \leq P_i$  (6/6)

- Demande en processeur dans le cas  $D_i \leq P_i$

$$C_p(0, L) = \sum_{i=1}^n \left( \left\lfloor \frac{L - D_i}{P_i} \right\rfloor + 1 \right) C_i$$

- Condition de faisabilité : condition **nécessaire et suffisante**
  - Pour tout L correspondant à une date d'échéance de tâches dans l'intervalle  $\min(Bp, \text{hyperpériode})$

$$\sum_{i=1}^n \left( \left\lfloor \frac{L - D_i}{P_i} \right\rfloor + 1 \right) C_i \leq L$$

Analyse d'ordonnabilité pour tâches périodiques  
Faisabilité ordonnancement à priorités - résumé

	$D_i = P_i$	$D_i \leq P_i$
Prio fixe	<b>Rate Monotonic</b> Faisabilité basée sur la charge (CS) $\sum_{i=1}^n \frac{C_i}{P_i} \leq n(2^{n-1})$	<b>Deadline Monotonic</b> Faisabilité basée sur la charge (CS) $\sum_{i=1}^n \frac{C_i}{D_i} \leq n(2^{n-1})$ Response time analysis (CNS) - <b>toute priorité fixe</b> $R_i = C_i + \sum_{j \in \text{hp}(i)} \left\lfloor \frac{R_i}{P_j} \right\rfloor C_j \leq D_i$
Prio dyn.	<b>EDF</b> Faisabilité basée sur la charge (CNS) $\sum_{i=1}^n \frac{C_i}{P_i} \leq 1$	<b>EDF</b> Processor demand approach (CNS) $C_p(0, L) = \sum_{i=1}^n \left( \left\lfloor \frac{L - D_i}{P_i} \right\rfloor + 1 \right) C_i$

Prise en compte du partage de ressources  
Introduction

- Hypothèses
  - Hypothèses initiales, sauf qu'on relâche l'hypothèse 4 d'indépendance entre tâches
- Ordonnements non préemptifs
  - Pas de problème (pas de préemption pendant les périodes d'utilisation des ressources)
- Ordonnements préemptifs à priorités
  - Identification pour chaque tâche  $T_i$  d'une borne supérieure sur sa durée de blocage par des tâches de priorité inférieure (**facteur de blocage**) :  $B_i$ 
    - Utilisation d'un protocole (PIP,PCP,SRP) **nécessaire**
  - Modification des conditions de faisabilité pour prise en compte des  $B_i$

Option STR – 2006-2007

205

Prise en compte du partage de ressources  
Ordonnements à priorités fixes : RM

- Condition de faisabilité d'origine (CS) ( $D_i = P_i$ )  $\sum_{i=1}^n \frac{C_i}{P_i} \leq n(2^{\frac{1}{n}} - 1)$
- Condition de faisabilité modifiée (CS)  $\forall i \in [1, n], \sum_{k=1}^i \frac{C_k}{P_k} + \frac{B_i}{P_i} \leq i(2^{\frac{1}{i}} - 1)$ 
  - Terme 1 : effet des préemptions par des tâches plus prioritaires
  - Terme 2 : blocage par des tâches moins prioritaires
- Condition plus simple, mais moins précise
  - On observe que  $\frac{B_i}{P_i} \leq \max_k (\frac{B_k}{P_k})$  et que  $i(2^{\frac{1}{i}} - 1) \leq n(2^{\frac{1}{n}} - 1)$
  - D'où la condition  $\sum_{i=1}^n \frac{C_i}{P_i} + \max(\frac{B_1}{P_1}, \dots, \frac{B_n}{P_n}) \leq n(2^{\frac{1}{n}} - 1)$

Option STR – 2006-2007

206

Prise en compte du partage de ressources  
Ordonnements à priorités fixes : DM

- Condition de faisabilité d'origine (CS) ( $D_i \leq P_i$ )

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

- Condition de faisabilité modifiée (CS)

$$\forall i \in [1, n], \sum_{k=1}^i \frac{C_k}{D_k} + \frac{B_i}{D_i} \leq i(2^{\frac{1}{i}} - 1)$$

- Condition plus simple, mais moins précise

$$\sum_{i=1}^n \frac{C_i}{D_i} + \max(\frac{B_1}{D_1}, \dots, \frac{B_n}{D_n}) \leq n(2^{\frac{1}{n}} - 1)$$

Option STR – 2006-2007

207

Prise en compte du partage de ressources  
Priorités fixes : extension de RTA

- Condition de faisabilité d'origine (CNS)

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_j}{P_j} \right\rceil C_j \leq D_i$$

- Condition de faisabilité modifiée

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_j}{P_j} \right\rceil C_j \leq D_i$$

- Contrairement à la condition d'origine, condition **suffisante** seulement (car  $B_i$  temps de blocage maximal pas nécessairement atteint)

Option STR – 2006-2007

208

Prise en compte du partage de ressources  
Ordonnancement à priorités dynamiques : EDF

- Condition de faisabilité d'origine (CNS) ( $D_i = P_i$ )

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

- Condition de faisabilité modifiée ( $D_i = P_i$ )

$$\forall i \in [1, n] \left( \sum_{k=1}^i \frac{C_k}{P_k} \right) + \frac{B_i}{P_i} \leq 1$$

Prise en compte du partage de ressources  
Ordonnancement à priorités dynamiques : EDF

- Condition de faisabilité d'origine (CS) ( $D_i \leq P_i$ )

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

- Condition de faisabilité modifiée ( $D_i = P_i$ )

$$\forall i \in [1, n] \left( \sum_{k=1}^i \frac{C_k}{D_k} \right) + \frac{B_i}{D_i} \leq 1$$

Prise en compte du partage de ressources  
Extension du processor demand

- Condition de faisabilité d'origine (CNS)

$$\forall L, C_p(0, L) = \sum_{i=1}^n \left( \left\lfloor \frac{L - D_i}{P_i} \right\rfloor + 1 \right) C_i \leq L$$

- Condition de faisabilité modifiée

$$\forall i, \forall L, C_p(0, L) = \sum_{k=1}^i \left( \left\lfloor \frac{L - D_k}{P_k} \right\rfloor + 1 \right) C_k + \left( \left\lfloor \frac{L - D_i}{P_i} \right\rfloor + 1 \right) B_i \leq L$$

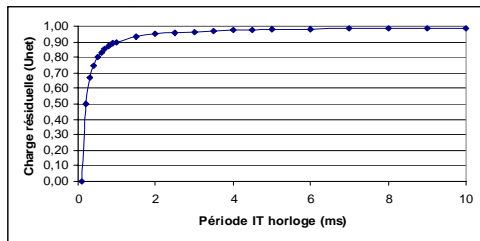
- Condition **suffisante** seulement

Prise en compte des coûts système  
Introduction

- Origine des coûts système
  - Traitements d'interruption (horloge, capteurs)
  - Appels système
  - Changements de contexte
- Supposés nuls jusqu'à présent
- Appels système
  - A intégrer dans les temps d'exécution des tâches ( $C_i$ )
  - Pose le problème d'évaluer les coûts des appels système
    - Mesures : attention aux conditions de mesures pour se placer dans le pire des cas (matériel : cache, pipelines, logiciel : état du système, chemin d'exécution, interruptions)
    - Analyse statique de code : voir partie sur l'obtention de WCET

Prise en compte des coûts système  
Prise en compte d'interruptions périodiques

- Exemple : interruption d'horloge
  - Ajout d'une tâche périodique
  - Diminue la charge processeur résiduelle (U<sub>net</sub>)
  - Exemple : IT horloge de 100μs



Option STR – 2006-2007

213

Prise en compte des coûts système  
Prise en compte des changements de contexte

- Estimer le nombre maximum de changements de contexte subis par chaque tâche (N<sub>i</sub>)
  - Dépendant de l'algorithme d'ordonnancement utilisé
- Estimer le temps d'un changement de contexte δ (sauv+restau)
  - Coût direct (sauvegarde/restauration registres par exemple)
  - Coût indirect : rechargement du cache/TLB/pipeline
- Intégrer dans condition de faisabilité, exemple pour RM

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

- Borne supérieure du nombre de préemptions subies par T<sub>i</sub> dans le cas de tâches périodiques (bornes plus fines peuvent être trouvées selon l'ordonnancement)

$$N_i = \sum_{k \in h(\tau_i)} \left\lceil \frac{P_i}{P_k} \right\rceil$$

Option STR – 2006-2007

214

Obtention des WCET (C<sub>i</sub>)

Plan

- Le WCET : définitions et éléments influençant le WCET
- Classes de méthodes d'obtention des WCET
  - Méthodes dynamiques
  - Méthode statiques
  - Comparaison
- Méthodes statiques d'obtention des WCET
  - Analyse de flot
  - Calcul des WCET
  - Analyse de bas niveau
  - L'analyseur Heptane (IRISA)
- Points ouverts et recherches actuelles

Option STR – 2006-2007

216

## WCET Définition

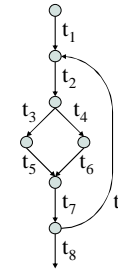
- ❑ **Borne supérieure** pour les temps d'exécution de portions de code
  - temps pris par le **processeur** pour exécuter cette portion de code
  - code considéré de manière **isolée**
  - **Attention** : WCET ≠ temps de réponse
- ❑ Défis pour l'obtention des WCET
  - Les estimations de WCET doivent 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

Option STR – 2006-2007

217

## WCET Éléments influençant le WCET

### Programme



- ❑ Mises en séquence possibles des actions du programme (chemins d'exécution)
  - Dépendent des données d'entrée
- ❑ Durée de chaque occurrence d'une action dans chaque chemin
  - Dépendant de l'architecture cible

Option STR – 2006-2007

218

## Méthodes d'estimation des WCET Méthodes dynamiques

- ❑ Principe
  - Jeu de données d'entrée au programme
  - Exécution (sur matériel ou simulateur)
  - Mesure
- ❑ Génération des jeux de test explicites
  - Définis par l'utilisateur
    - Nécessité d'une expertise des programmes (sujet aux erreurs)
  - Exhaustifs
    - Possible uniquement quand les entrées ont des domaines finis
    - Problème de combinatoire
  - Générés automatiquement pour maximiser le temps d'exécution
    - Algorithmes de type génétique ou recuit simulé
  - Pas de garantie d'être supérieur à tout temps d'exécution : problème de sûreté

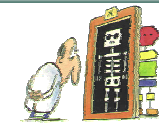


Option STR – 2006-2007

219

## Méthodes d'estimation des WCET Méthodes statiques

- ❑ Principe
  - Analyser de la structure du programme (**pas d'exécution**) au niveau source et/ou objet
  - Calculer la WCET à partir de la structure
- ❑ Composants de l'analyse
  - Analyse de flot
    - Détermine les chemins d'exécutions possibles
  - Analyse de bas niveau
    - Détermine le temps d'exécution d'une séquence d'instructions sur une architecture donnée (dépendant du matériel)
  - Calcul
    - Calcul du WCET à partir des composants précédents
    - Par construction, tous les chemins sont considérés : l'analyse est sûre



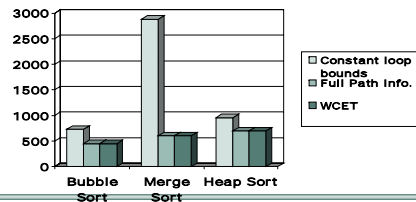
Option STR – 2006-2007

220



Méthodes statiques  
Analyse de flot (3/3)

- ❑ Modes de détermination des flots
  - Automatique : infaisable dans le cas général (halting problem)
  - Manuelle : annotations
    - Simples constantes sur les boucles
    - Annotations symboliques (ex: IRISA)
- ❑ Résultats de l'analyse de flot (P. Puschner)

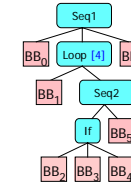


Option STR – 2006-2007

225

Méthodes statiques : calcul  
Analyse à base d'arbres (tree-based)

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



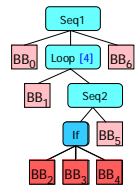
- ❑ Principe de la méthode
  - Détermination des temps d'exécution des blocs de base (analyse de bas-niveau)
  - Calcul des temps d'exécution des chacune des structures syntaxiques du langage en utilisant les temps des blocs de base et un timing schema pour chaque construction

Option STR – 2006-2007

226

Méthodes statiques : calcul  
Analyse à base d'arbres (tree-based)

WCET(SEQ)	S1...Sn	WCET(S1) + ... + WCET(Sn)
WCET(IF)	if(test) then else	WCET(test) + max(WCET(then), WCET(else))
WCET(LOOP)	for(,tst;inc) (body)	maxiter * (WCET(tst)+WCET(body+inc)) + WCET(tst)



Système d'équations

$$\begin{aligned}
 WCET(Seq1) &= WCET\_BB0 + WCET(Loop) + WCET\_BB\_6 \\
 WCET(Loop) &= 4 * (WCET\_BB1 + WCET(Seq2)) + WCET\_BB1 \\
 WCET(Seq2) &= WCET(If) + WCET\_BB5 \\
 WCET(If) &= WCET\_BB2 + \max(WCET\_BB3, WCET\_BB4)
 \end{aligned}$$

Option STR – 2006-2007

227

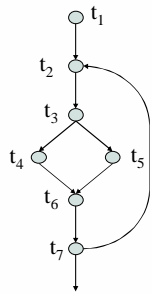
Méthodes statiques : calcul  
Analyse à base d'arbres (tree-based)

- ❑ Attrait des méthodes tree-based
  - Lien avec le code source aisé
  - Complexité des calculs maîtrisée
  - Mais, ...
- ❑ Limitations
  - Ne supporte pas toutes les optimisations de compilation
  - Prise en compte d'informations de flot élaborées n'est pas aisée (si non conforme à l'arbre syntaxique)

Option STR – 2006-2007

228

Méthodes statiques : calcul  
Analyse IPET (Implicit Path Enumeration Technique)



- Programmation linéaire en nombres entiers
  - But
    - max:  $f_1 t_1 + f_2 t_2 + \dots + f_n t_n$
  - Contraintes structurelles
    - $\forall v, \sum_{e \in \text{In}(v)} f_{a_i} = \sum_{e \in \text{Out}(v)} f_{a_i} = f_v$
  - Contraintes sur les chemins
    - exemples :  $f_i \leq k$  (maxiter boucle)
    - $f_i + f_j \leq 1$  (chemins mutuellement exclusifs)
    - $\sum c_i f_i \leq k$  (ratio entre nombres d'exécutions)
- Programmation par contraintes
  - Expression de contraintes pour exprimer des relations plus complexes sur les chemins

Option STR – 2006-2007

229

Méthodes statiques : calcul  
Analyse IPET

- Attrait des méthodes IPET
  - Travaille sur une structure de bas niveau, donc supporte les flots non structurés (goto, jumps, ...)
  - Supporte toutes les optimisations de compilation
  - Mais, ...
- Limitations
  - Complexité des calculs non maîtrisée
  - Liens avec le code source non évidents (maîtrise des optimisations de compilation)
  - Pas de retour explicite du pire chemin d'exécution

Option STR – 2006-2007

230

Méthodes statiques : analyse de bas niveau  
Introduction

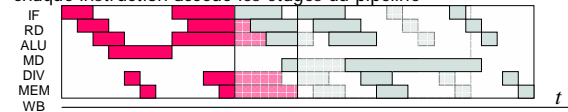
- Architecture simple
  - Temps d'exécution d'une instruction dépend uniquement de son type et de ses opérandes
  - Pas de recouvrement entre instructions, pas de hiérarchie de mémoire
- Architecture complexe
  - Effets locaux
    - Recouvrement entre instructions (pipelines)
  - Effets globaux
    - Caches de données, d'instructions, prédicteurs de branchement
    - Demande une connaissance de l'ensemble du code

Option STR – 2006-2007

231

Méthodes statiques : analyse de bas niveau (locale)  
Prise en compte des pipelines

- Principe : parallélisme entre instructions successives (effet local)
- Problème : simple ajout des temps d'exécution trop pessimiste
- Une solution (intra-BB) : tables de réservation décrivant quand chaque instruction accède les étages du pipeline



- Obtenus par l'analyseur lui-même ou outil externe (simulateur, processeur cible)
- Modification de la méthode de calcul
  - Tree-based: opérateur d'addition spécifique
  - IPET: contraintes supplémentaires dans le problème d'ILP

Option STR – 2006-2007

232

Méthodes statiques : analyse de bas niveau (globale)  
Caches d'instructions

- ❑ Cache
  - Tire profit de la localité spatiale et temporelle des accès aux instructions
  - **Spéculatif** : comportement dépend du comportement passé des programmes
  - Bon comportement en moyenne, mais problème de déterminisme en contexte temps-réel strict
- ❑ Problème : estimation **sûre** du comportement des caches
  - Solution simple (tout miss) est excessivement pessimiste
  - Objectif : prédire si une instruction causera un **hit** (de manière sûre) ou pourra causer un **miss** (de manière conservatrice)

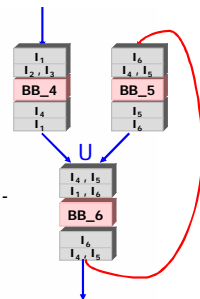
Méthodes statiques : analyse de bas niveau (globale)  
Caches d'instructions : simulation statique de cache

- ❑ Estimation **sûre** du comportement des caches
  - Prédit si une instruction causera un **hit** (de manière sûre) ou pourra causer un **miss** (de manière conservatrice)
- ❑ Calcul d'**états abstraits de caches** (ACS)
  - État du cache représentant **tous** les chemins d'exécution possibles
  - Itération de point fixe pour le calcul
- ❑ Division des instructions en catégories selon les ACS
  - *always hit*
  - *always miss*
  - *first miss, first hit* (cas d'instructions dans des boucles)

Méthodes statiques : analyse de bas niveau (globale)  
Caches d'instructions : simulation statique de cache

```

input_state(top) = all invalid lines
while any change do
  for each basic block instance B do
    input_state(B) = null
    for each immediate predecessor P of B do
      input_state(B) += output_state(P)
    end for
    output_state(B) = (input_state(B)+ prog_lines(B)) -
      conf_lines(B)
  end for
end while
    
```



Amélioration : prise en compte des niveaux d'imbrication des boucles

Méthodes statiques : analyse de bas niveau (globale)  
Caches d'instructions - gel de caches

- ❑ Limites de l'analyse de caches
  - Degrés d'associativité élevés, caches totalement associatifs
  - Politiques de remplacement de caches non déterministes
    - Dégradation des performances des méthodes d'analyse
  - Latence lors des changements de contexte
- ❑ **Gel** du contenu du cache (**cache locking**) – contrôle contenu par logiciel
  - Rend le temps d'accès à chaque information déterministe
  - Pas d'impact sur les temps de changement de contexte
  - Disponible dans de nombreuses architectures (PowerPC, Coldfire, ...)
  - Obtention des WCETs simplifiée, indépendamment de la politique de remplacement de cache
  - Gel partiel (MPC 7451) : cohabitation applications temps-réel strict et souple

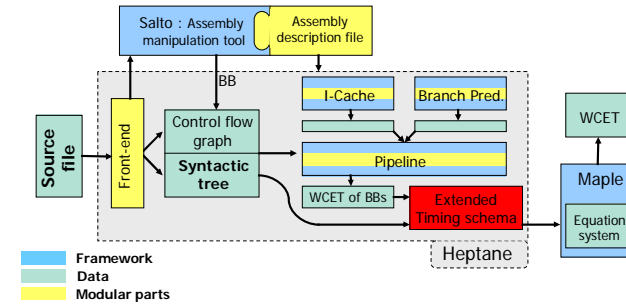
Méthodes statiques : analyse de bas niveau (globale)  
Autres éléments à prendre en compte

- Caches de données
  - Problème : obtention de l'adresse des données (dynamique)
  - Une solution : exécution symbolique [Chalmers]
  - Extension d'un simulateur de processeurs à des données inconnues
    - Exemple : cas d'un branchement
      - Valeur testée connue : on connaît le flot d'exécution
      - Valeur testée inconnue : on explore les deux branches
    - Système de fusion de chemins pour éviter de dérouler totalement les boucles
- Prédicteurs de branchement
  - Locaux (basé sur historique dernières prédictions du branchement) [IRISA]
  - Globaux (basé sur historique du branchement + autres branchements exécutés récemment) [Singapour] : Mapping vers un problème d'ILP

Option STR – 2006-2007

237

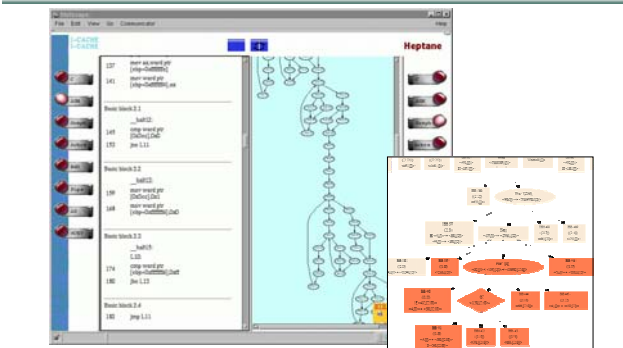
Méthodes statiques  
Heptane



Option STR – 2006-2007

238

Méthodes statiques  
Heptane



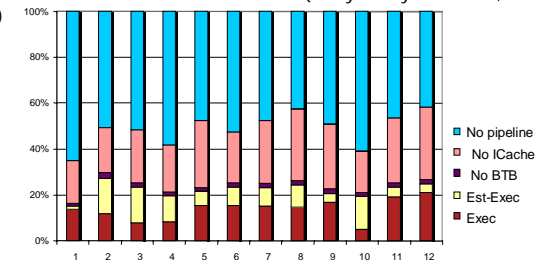
Option STR – 2006-2007

239

Méthodes statiques

Heptane : quelques résultats quantitatifs

- Impact de la modélisation d'architecture (analyse noyau RTEMS, Irisa)



- Sans prise en compte architecture, facteur de surestimation = 12.5
- Prise en compte architecture réduction pessimisme d'un facteur 6.6

Option STR – 2006-2007

240

## Points ouverts et recherches actuelles

- Recherches actuelles : analyse de haut niveau
  - Détermination automatique des flots (Målardalen)
  - Paradigmes de programmation pour le temps-réel facilitant le déterminisme (donc l'obtention des WCET) : single path paradigm (Vienne)
- Recherches actuelles : analyse de bas niveau
  - Vers une prise en compte complète de matériel complexe
    - exécution spéculative, prédicteurs de branchement
    - autres éléments à traiter : traduction d'adresse, DMA, rafraîchissement DRAM, architectures SMT, ...
  - Vers une utilisation prévisible de matériel complexe : gel de cache
  - Vers la conception de matériel prévisible
- Recherches actuelles : vers une meilleure précision des WCET au détriment de la sûreté : Approches probabilistes (York)

Option STR – 2006-2007

241

## Analyse d'ordonnançabilité Bilan

- De nombreux résultats théoriques en ordonnancement temps-réel
  - Difficile d'en faire le tour en peu de temps (400 pages consacrées au sujet dans le livre de G. Butazzo par exemple)
  - Ici, on n'a présenté que les principaux résultats dans les cadres les plus simples
- Attention aux hypothèses d'application des conditions de faisabilité
  - Exemple : application de la condition sur la charge  $\sum_{i=1}^n \frac{C_i}{P_i} \leq n(2^n - 1)$  à des tâches qui ne sont pas indépendantes ou ont un Ci sous estimé
- Utiliser autant que possible les résultats théoriques existants (**degré de confiance** par rapport aux méthodes de test, utilisables dans les **phases préliminaires** du développement)
- Peu d'outils commerciaux utilisant ces résultats

Option STR – 2006-2007

242

## Ordonnancement hybride de tâches temps-réel strict et non strict

## Ordonnancement hybride Introduction (1/2)

- Résultats présentés précédemment
  - Tâches périodiques ou tâches à dates d'arrivée sont connues a priori
  - Contraintes temps-réel strictes
- Caractéristiques typiques des tâches dans les systèmes temps-réel
  - Tâches de **contrôle** : critiques, échéances strictes, lancement commandé par le temps (**time-driven**) : arrivées périodiques ou à dates connues
  - Autres tâches : arrivées commandées par des événements (**event-driven**) sporadiques ou aperiodiques, contraintes fermes, souples ou sans contraintes de temps
  - 2<sup>ème</sup> type de tâches pas supporté par les résultats du chapitre précédent
- Remarque : on ne peut pas garantir les tâches non périodiques hors-ligne, sauf si délai inter-arrivée minimum connu (sporadiques)
- Échéance **fermes (firm)** : tâches dont on veut garantir les échéances, même si la garantie est fournie en-ligne uniquement

Option STR – 2006-2007

244

## Ordonnancement hybride Introduction (2/2)

- Ordonnancement de jeux de tâches **hybrides**
  - Ensemble de tâches **périodiques** temps-réel **strict**
  - Ensemble de tâches **apériodiques** temps-réel **ferme** ou **souple**
- Hypothèses considérées dans le chapitre
  - Tâches périodiques  $T_i$  synchrones à échéance sur requête (n tâches de période  $P_i$  et de WCET  $C_i$ )
  - Ordonnancement des tâches périodiques par un ordonnancement à priorités fixes (sauf mention contraire, RM)
  - Dates d'arrivée des tâches apériodiques  $J_a$  inconnues a priori

Option STR – 2006-2007

245

## Ordonnancement hybride Plan du chapitre

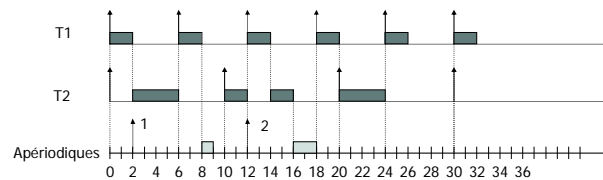
- Tâches apériodiques en tâche de fond (**background scheduling**)
- Gestion des tâches apériodiques par un serveur
  - Traitement par scrutation (**polling server**)
  - Serveur ajournable (**deferrable server**)
  - Echanges de priorités (**priority exchange server**)

Option STR – 2006-2007

246

## Background scheduling (1/3)

- Principe
  - Exécution des tâches apériodiques quand aucune tâche périodique n'est prête à s'exécuter
- Exemple
  - $T_1 : C_1 = 2 \quad P_1 = 6$
  - $T_2 : C_2 = 4 \quad P_2 = 10$

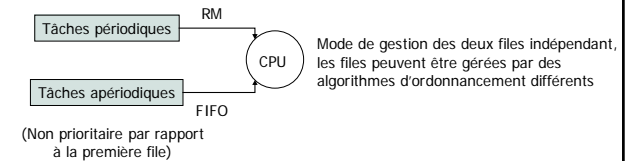


Option STR – 2006-2007

247

## Background scheduling (2/3)

- Remarque : par construction, aucun impact sur les tâches périodiques
- Mise en œuvre : simple



Option STR – 2006-2007

248

### Background scheduling (3/3)

- Intérêt
  - Simplicité de la mise en œuvre
  - Pas d'impact sur les tâches périodiques (conditions de faisabilité inchangées)
- Limitations
  - Capacité processeur attribuée aux tâches apériodiques dépendante de la charge due aux tâches périodiques
    - Le temps de réponse des tâches apériodiques peut être long à charge importante
  - Utilisable à charge modérée uniquement, pour des tâches apériodiques non urgentes

Option STR – 2006-2007

249

### Ordonnancement hybride à base de serveurs Serveur par scrutation (polling server) (1/8)

- Principe
  - Utilisation d'un **serveur** : tâche périodique dédiée à l'exécution des tâches apériodiques (principe commun à tous les ordonnanceurs à base de serveurs)
    - Tâche caractérisée par  $(C_s, P_s)$ ,  $C_s$  nommé **capacité** du serveur
    - Tâches apériodiques exécutées pendant au maximum  $C_s$  unités de temps toutes les périodes  $P_s$
  - Code du serveur (polling server uniquement)
    - Servir les requêtes apériodiques dans la limite de la capacité  $C_s$  (ordre de service indépendant des tâches périodiques)
    - Si pas de requête apériodique en attente de service, suspension jusqu'à la période suivante

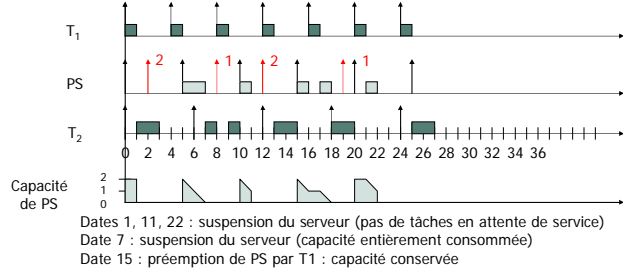
Option STR – 2006-2007

250

### Ordonnancement hybride à base de serveurs Polling server (2/8)

- Exemple de fonctionnement (ordonnancement RM)
 

$T_1 : C_1 = 1 \quad P_1 = 4 \quad T_2 : C_2 = 2 \quad P_2 = 6$   
 Serveur :  $C_s = 2 ; T_s = 5$  (priorité intermédiaire entre  $T_1$  et  $T_2$ )



Option STR – 2006-2007

251

### Ordonnancement hybride à base de serveurs Polling server (3/8)

- Condition de faisabilité pour les tâches périodiques
  - Condition modifiée (prise en compte de l'exécution du serveur)
  - Condition suffisante de faisabilité sous RM

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

- Extension à  $m$  polling servers de priorités différentes pour tâches apériodiques d'importance différentes

$$\sum_{i=1}^n \frac{C_i}{P_i} + \sum_{j=1}^m \frac{P_j}{C_j} \leq (n+m)(2^{\frac{1}{n+m}} - 1)$$

Option STR – 2006-2007

252

Ordonnancement hybride à base de serveurs  
Polling server (4/8)

- Condition pour la garantie d'une tâche aperiodique (échéance ferme)
    - A. Condition pour une tâche **isolée** (i.e. pas d'autre tâche aperiodique en attente)  $J_a$ , arrivant en  $A_a$ , de WCET  $C_a$  et d'échéance relative  $D_a$ 
      - Temps de prise en compte maximal de la tâche :  $P_s$  (car tâche isolée)
      - Si  $C_a \leq C_s$ , l'exécution de  $J_a$  est terminée au plus tard en  $2 P_s$  et la condition d'acceptation est  $2 P_s \leq D_a$
      - Dans le cas général ou on n'a pas  $C_a \leq C_s$ , l'exécution de  $J_a$  est terminée au plus tard en
 
$$P_s + \left\lceil \frac{C_a}{C_s} \right\rceil P_s$$
- et la condition d'acceptation est alors  $P_s + \left\lceil \frac{C_a}{C_s} \right\rceil P_s \leq D_a$

Ordonnancement hybride à base de serveurs  
Polling server (5/8) - exercice 14

- Remarque
  - La condition d'acceptation (garantie) d'une tâche aperiodique est une condition **suffisante** seulement, car on ne sait pas quand le serveur s'exécute par rapport au début de sa période (dépend de sa priorité)
- Exercice 14
  - Etablir une condition nécessaire et suffisante d'acceptation d'une tâche aperiodique isolée  $T_a$ , arrivant en  $A_a$ , de WCET  $C_a$  et d'échéance relative  $D_a$  dans le cas où le **polling server est la tâche la plus prioritaire** (i.e. s'exécute toujours en début de période)
  - Solution

Ordonnancement hybride à base de serveurs  
Polling server (6/8)

- Condition de garantie d'une tâche aperiodique : **tâches non isolées**, serveur de **plus haute priorité**
  - Tâches aperiodiques à échéances fermes stockées et exécutées par échéances croissantes (EDF)
  - Pour tout temps  $t$ , la quantité totale de calcul devant être servie dans un intervalle  $[t, t+D_k]$  est la somme des temps résiduels à exécuter pour les tâches d'échéance  $D_i \leq D_k$ 

$$C_{ape} = \sum_{i=1}^k res_i(t)$$
  - Soit  $C_s(t)$  la capacité résiduelle du serveur à l'instant  $t$ . Comme PS a la plus haute priorité, une portion de  $C_{ape}$  égale à  $C_s(t)$  est exécutée immédiatement.

Ordonnancement hybride à base de serveurs  
Polling server (7/8)

- Date de fin d'une requête  $J_k$ 
    - $F_k = t + C_{ape}(t, D_k)$  si  $C_{ape}(t, D_k) \leq C_s(t)$
    - $F_k = (N_k + G_k) P_s + Res_k$  sinon
- avec  $N_k$  et  $G_k$  définis comme dans l'exercice précédent
- $$N_k = \left\lfloor \frac{C_{ape}(t, D_k) - C_s(t)}{C_s} \right\rfloor$$
- $$G_k = \left\lfloor \frac{t}{P_s} \right\rfloor$$
- $$Res_k = C_{ape}(t, D_k) - C_s(t) - N_k C_s$$
- Condition d'acceptation d'une tâche aperiodique :  $\forall k \in [1, n_a]$ , avec  $n_a$  le nombre de tâches aperiodiques, on vérifie que  $F_k \leq D_k$
  - Remarque : le temps d'acceptation est en  $O(n_a)$  avec  $n_a$  le nombre de tâches aperiodiques en cours d'exécution

Ordonnancement hybride à base de serveurs  
Polling server (8/8)

- Avantages
  - Reste simple, au niveau :
    - Des conditions de faisabilité pour les tâches périodiques
    - De l'acceptation en-ligne des tâches aperiodiques
  - Meilleur service pour les tâches aperiodiques que la gestion des tâches aperiodiques en tâches de fond
- Inconvénients
  - Capacité du serveur perdue en cas d'absence de tâche aperiodique en attente, jusqu'à la période suivante

Option STR – 2006-2007

257

Ordonnancement hybride à base de serveurs  
Serveur ajournable (Deferrable Server) (1/5)

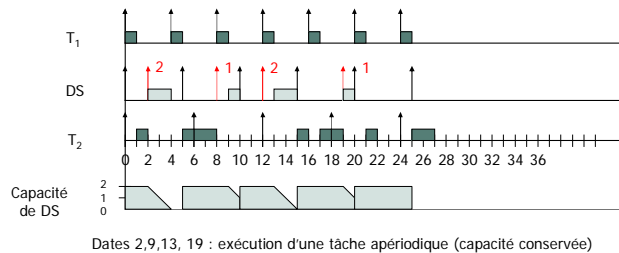
- Principe
  - Serveur dédié à l'exécution des tâches aperiodiques (idem au polling server, PS)
  - Différence avec PS :
    - Quand le serveur n'a pas de tâche aperiodique en attente, on **conserve sa capacité courante** jusqu'à la fin de la période pour le traitement des tâches aperiodiques

Option STR – 2006-2007

258

Ordonnancement hybride à base de serveurs  
Deferrable Server (2/5)

- Exemple de fonctionnement (même exemple que pour PS)



Option STR – 2006-2007

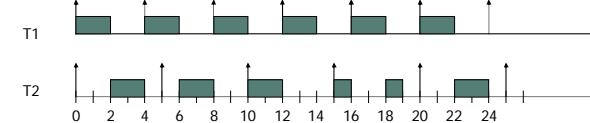
259

Ordonnancement hybride à base de serveurs  
Deferrable Server (3/5)

- Condition de faisabilité pour les tâches périodiques
  - Le serveur DS n'est **plus équivalent à une tâche périodique** vis à vis de la faisabilité du système (il **ajourne** son exécution, ce que ne font pas les tâches périodiques)
  - La condition suffisante pour l'ordonnancement RM **n'est plus applicable**

- Illustration

- Cas a. Deux tâches périodiques T1 et T2 ( $C_1 = C_2 = 2$ ,  $P_1 = 4$ ,  $P_2 = 5$ )



(système ordonnancement sous RM)

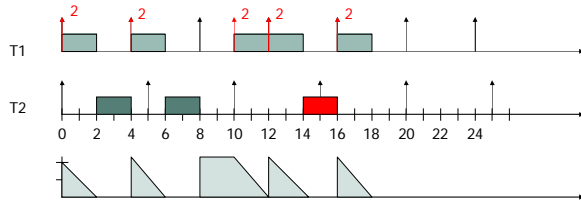
Option STR – 2006-2007

260

Ordonnancement hybride à base de serveurs  
Deferrable Server (4/5)

□ Illustration (suite)

- Cas b.  $T_1$  remplacé par un DS de WCET et périodes équivalentes



$T_2$  rate son échéance à la date 16

Option STR – 2006-2007

261

Ordonnancement hybride à base de serveurs  
Deferrable Server (5/5)

□ Condition de faisabilité pour les tâches périodiques

- En notant  $U_s = (C_s/P_s)$ , le système est faisable si la condition suivante est vérifiée (condition suffisante)

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

□ Condition d'acceptation des tâches aperiodiques

- Existe
- Même complexité algorithmique que PS
- Accepte plus de tâches aperiodiques que PS

Option STR – 2006-2007

262

Ordonnancement hybride à base de serveurs  
Serveur à échange de priorités (Priority Exchange Server) (1/3)

□ Principe

- Serveur dédié à l'exécution des tâches aperiodiques (idem PS et DS)
  - Par la suite, on considère que le serveur a la plus haute priorité
- Quand le serveur a une tâche aperiodique a exécuter, l'exécute à la priorité du serveur (idem PS et DS)
- Quand le serveur n'a pas de tâche aperiodique en attente (différent PS et DS) : **échange de priorité** avec la tâche périodique prête (+ prio)
  - Exécution d'une tâche périodique **à la priorité du serveur**
  - Accumulation d'une capacité équivalente, **à la priorité de la tâche périodique**. Capacité conservée jusqu'à la fin de l'exécution de la tâche périodique, perdue ensuite

Option STR – 2006-2007

263

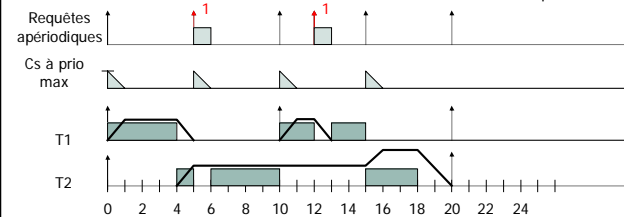
Ordonnancement hybride à base de serveurs  
Priority exchange server (2/3)

□ Exemple de fonctionnement

$T_1 : C_1 = 4 \quad P_1 = 10$                        $T_2 : C_2 = 8 \quad P_2 = 20$

Serveur :  $C_s = 1 ; T_s = 5$  (plus haute priorité)

Cs pour une priorité donnée



Option STR – 2006-2007

264

Ordonnancement hybride à base de serveurs  
Priority exchange server (3/3)

- Condition de faisabilité pour les tâches périodiques

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

Borne plus haute que pour DS  
(sauf pour  $U_s$  très faible)

- Acceptation de tâches apériodiques beaucoup plus complexe (calcul du temps de fin, à cause de l'utilisation de priorités différentes)

Option STR – 2006-2007

265

Ordonnancement hybride dans le cadre priorités fixes  
Bilan (1/2)

- Non existence de serveur optimaux
  - Théorème (Tia-Liu-Shankar). Pour tout ensemble formé :
    - de tâches périodiques gérés par un ordonnancement à priorités fixes
    - de tâches apériodiques gérées selon un ordonnancement quelconque
 il n'existe pas d'algorithme qui minimise le temps de réponse de toutes les tâches apériodiques
- Remarque : il existe des ordonnancements hybrides similaires dans le cas de tâches périodiques ordonnancées par des algorithmes à priorités dynamiques (e.g. EDF)

Option STR – 2006-2007

266

Ordonnancement hybride dans le cadre priorités fixes  
Bilan (2/2)

	Performance	Temps de calcul	Besoins mémoire	Complexité implem.
Background	●	●	●	●
Polling server	●	●	●	●
Deferrable server	●	●	●	●
Priority exchange server	●	●	●	●

Option STR – 2006-2007

267

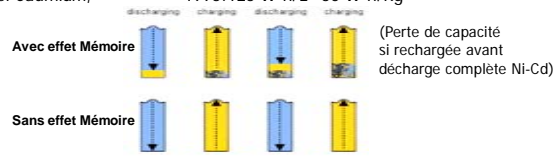
Minimisation de la consommation énergétique à l'aide du système d'exploitation

Crédits : la quasi-intégralité de ces transparents a été généreusement léguée par Gilbert Cabillic (INRIA/IRISA)

## Introduction

### Motivations

- ❑ Systèmes embarqués
  - Certains systèmes embarqués sont autonomes en énergie (ordinateurs portables, agendas électroniques, téléphones portables)
  - Utilisation de batteries : sources d'énergie épuisables
- ❑ Caractéristiques des batteries [Fuj97][Bel01]
  - Lithium ion, 1997: 380 W-h/L 135 W-h/Kg
  - Nickel Metal Hydride, 1995: 150 W-h/L 50 W-h/Kg
  - Nickel Cadmium, 1995: 125 W-h/L 50 W-h/Kg



Option STR – 2006-2007

269

## Introduction

### Niveaux

- ❑ La gestion de l'énergie peut être faite à plusieurs niveaux



- ❑ Plus le niveau est élevé, plus le gain sera important (car la connaissance des informations est plus grande) [SRC84]
- ❑ Mais
  - L'utilisateur n'a pas la capacité de prendre des décisions sur des délais très courts
  - Dans un contexte multi-application, les décisions peuvent être contradictoires (vue locale seulement)
- ❑ Donc, le système d'exploitation est bien placé pour réaliser la gestion de l'énergie

Option STR – 2006-2007

270

## Introduction

### Evaluation stratégique (1/2) Utilisation composants [Lorch01]

Component	Hypo- thetical 386	Duo 230	Duo 270c	Duo 280c	Average
Processor	4%	17%	9%	25%	14%
Hard disk	12%	9%	4%	8%	8%
Backlight	17%	25%	26%	25%	23%
Display	4%	4%	17%	10%	9%
Modem	n/a	1%	0%	5%	2%
FPU	1%	n/a	3%	n/a	2%
Video	26%	8%	10%	6%	13%
Memory	3%	1%	1%	1%	2%
Other	33%	35%	28%	22%	30%
Total	6 W	5 W	4 W	8 W	6 W

- ❑ Réseau sans fil : consommation de 100mW à 1W selon le réseau

Option STR – 2006-2007

271

## Introduction

### Evaluation d'une stratégie (2/2)

- ❑ Relativiser l'importance de la stratégie (considérer le problème de manière globale)
  - Exemple :
    - Une stratégie permet 50% de réduction d'énergie pour l'exécution du modem
    - Si le modem correspond à 4% d'utilisation globale du système embarqué, il y aura seulement 2% de gain au final
- ❑ Autres paramètres à considérer
  - Temps d'exécution (temps-réel ou performance)
    - Exemple : à quoi sert de diminuer de 50% la consommation en énergie si on augmente les temps d'exécution ?
  - Coût
  - Existence de conflits lors de l'optimisation des différents paramètres : besoin de compromis

Option STR – 2006-2007

272

## Introduction

Différentes classes de stratégies

### 1. Transition

- Passage d'un mode de fonctionnement à un autre (plus économe en énergie)
  - Exemple: mise en veille du processeur

### 2. Load-change

- Modification de la charge d'un composant matériel
  - Exemple : mémoire cache de disque (diminution charge du disque)

### 3. Adaptation

- Adaptation d'un composant en utilisant des techniques nouvelles
  - Exemple: remplacement d'un disque dur par une mémoire flash

Option STR – 2006-2007

273

## Plan

- Introduction
- Les stratégies du système d'exploitation
  - concernant le support de stockage
  - concernant l'écran
  - concernant le processeur
  - concernant les périphériques de communication sans fil
- Conclusion

Option STR – 2006-2007

274

## Support de stockage

Caractéristiques de quelques disques durs

### □ Typiquement, 5 niveaux de fonctionnement (conso décroissante)

- Actif : en positionnement / lecture / écriture
- Idle : pas de positionnement / lecture / écriture, mais rotation active
- Standby : arrêt moteur, mais électronique du contrôleur active, cache conservé
- Sleep : arrêt moteur et élect. du contrôleur (sauf détect. reset), cache perdu
- Off

Hard disk	Toshiba MK3017GAP	IBM Travel- star 48GH	Fujitsu MHL2300AT	Hitachi DK22AA-18
Capacity	30 GB	48 GB	30 GB	18 GB
Idle power	0.7 W	0.9 W	0.85 W	0.8 W
Standby power	0.3 W	0.25 W	0.28 W	0.25 W
Sleep power	0.1 W	0.1 W	0.1 W	0.125 W
Spin-up time	4 sec	1.8 sec	5 sec	3 sec
Spin-up energy	10.8 J	9.0 J	22.5 J	13.5 J

Temps de redémarrage du moteur après période sleep/standby

Option STR – 2006-2007

275

## Support de stockage

Caractéristiques des mémoire flash

- Mémoire non volatile (sans consommation d'énergie)
- Contraintes
  - Techniquement, support Read only
  - Pour faire une écriture, il faut détruire et réécrire un segment complet à chaque fois
    - Taille segment : 0.5-128 KB, 15 µsec/octet pour détruire le segment
  - Un segment peut être détruit un nombre limité de fois (100000 à 1.000000)
- Coût : 1MB= (de 1\$ à 3\$), soit **125-450 fois plus cher qu'un disque dur** et **8-24 fois plus cher** que de la mémoire externe DRAM
- Caractéristiques
  - Consommation Read/Write: 0.15 W – 0.47 W (faible / disque)
  - Read Speed: 85 ns per byte (idem DRAM)
  - Write Speed: 4-10 µsec (**10-100 fois plus lent qu'un disque dur**)

Option STR – 2006-2007

276

## Support de stockage

Transition - Mise en sommeil/Arrêt du disque (1/2)

- Plusieurs approches
  - Mettre le disque en sommeil - **mode sleep** (le plus utilisé)
    - L'énergie gagnée est plus importante
    - Le temps de redémarrage du disque est quasiment identique (dominé par le temps de redémarrage du moteur)
    - Le mode standby est une caractéristique récente des disques
  - Mettre le disque en mode standby
    - Intérêt / sleep : on conserve le cache (plus ou moins intéressant selon si on a en plus un cache en mémoire)
    - Désavantage / sleep : gain en énergie moins important
  - Arrêter le fonctionnement du disque

Option STR - 2006-2007

277

## Support de stockage

Transition - Mise en sommeil du disque (2/2)

- Mise en sommeil après une période d'inactivité **fixe**
  - Les constructeurs : 3-4 minutes
  - Plus petits seuils : étudié dans [LKHA94] : seuils de 1-10 secondes
    - gain en énergie double par rapport à un seuil de 3-5 mn, mais
    - délai de redémarrage (8-30 s par heure, perceptible)
    - durée de vie du disque raccourcie
- Mise en sommeil après une période d'inactivité **variable**
  - [KLV95] définit :
    - un ensemble de seuils possibles
    - une politique de surveillance de l'utilisation du disque afin de choisir le seuil approprié

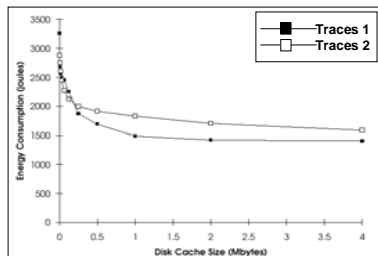
Option STR - 2006-2007

278

## Support de stockage

Load-change - Changement de la charge du disque (1/2)

- Données : **introduction d'un cache** [LKHA94]
  - Cache de 1MB -> 50% de réduction.
  - Au delà d'un certain seuil, pas de gain (idem cache mémoire)



Option STR - 2006-2007

279

## Support de stockage

Load-change - Changement de la charge du disque (2/2)

- Noms de fichiers et attributs : **introduction d'un cache**
  - Cache de 50 KB -> 17% de réduction [LKHA94]
- Problème posé sur le swap (pagination mémoire)
  - Identifier les bonnes pages qui permettent de minimiser les transferts mémoire depuis/vers disque
  - Remarque : optimisation pour le temps d'exécution et l'énergie ne sont pas contradictoires ici

Option STR - 2006-2007

280

## Support de stockage

Adaptation - Utilisation d'une mémoire flash comme cache disque

- ❑ Dû à la performance des écritures, on place la mémoire flash comme un cache L2, entre le cache disque DRAM et le disque.
- ❑ [MDK93]
  - Taille de Cache de 1-40 MB permet de minimiser de 20-40 % d'énergie et accélérer les accès de 30-70 %
- ❑ Problème
  - Nécessité de maintenir une liste des segments écrits afin de répartir les écritures sur la mémoire flash afin de ne pas perdre les données (nombre de réécritures limité)

Option STR – 2006-2007

281

## Support de stockage

Adaptation - Utilisation d'une mémoire flash comme disque

- ❑ Gestion différente d'un disque dur
  - Utilisation d'un journal des écritures afin de minimiser le nombre de destruction de segments [KNM95]
  - Réaliser des permutations de segments afin de ne pas réaliser les écritures toujours sur le même segment (nombre écritures limité)
- ❑ Réduction énergétique de 60-90 % par rapport à un disque dur pour une performance similaire [DKM94], mais durée de vie limitée

Option STR – 2006-2007

282

## Support de stockage

Adaptation - Utilisation du réseau sans fil comme un disque distant

- ❑ Limitations
  - La faible bande passante du réseau ne permet pas d'atteindre des débits importants
  - Certains réseaux sans fil ont des consommations élevées

Option STR – 2006-2007

283

## Plan

- ❑ Introduction
- ❑ Les stratégies du système d'exploitation
  - concernant le support de stockage
  - concernant l'écran
  - concernant le processeur
  - concernant les périphériques de communication sans fil
- ❑ Conclusion

Option STR – 2006-2007

284

## Périphérique d'affichage

- ❑ Duo280 (Apple powerbook)
  - Périphérique d'affichage écran: 0.75W
  - Lumière de fond: 3.40W
- ❑ Stratégie de transition
  - Extinction de la lumière de fond au bout d'un certain temps
    - 32-67% de gain [Lor95a]
- ❑ Stratégies d'adaptation
  - Utilisation d'un périphérique détectant la luminosité externe afin d'utiliser la puissance d'éclairage la plus faible [Soh95][IPAQ spec]
  - Travail sur les écrans / luminosité

Option STR – 2006-2007

285

## Plan

- ❑ Introduction
- ❑ Les stratégies du système d'exploitation
  - concernant le support de stockage
  - concernant l'écran
  - concernant le processeur
  - concernant les périphériques de communication sans fil
- ❑ Conclusion

Option STR – 2006-2007

286

## Processeur

Transition - stratégies de mise en sommeil du processeur (1/3)

- ❑ Modes de fonctionnement typiques du processeur
  - Actif
  - En sommeil
  - Tension d'alimentation réduite conjointement avec fréquence réduite (DVS, Dynamic Voltage Scaling)
  - Arrêt
- ❑ Exemple : Intel's Mobile Pentium III
  - 7 états (triés par niveau de consommation décroissant) : Normal, Stop Grant, Auto Halt, Quick Start, Halt/Grant Snoop, Sleep, Deep sleep
- ❑ Délai de retour en mode actif dépendant du "niveau de sommeil"
  - Exemple pour l'Intel's Mobile Pentium III : 30  $\mu$ s à partir du mode *Deep sleep*, 10 cycles de bus à partir du mode *Auto halt*

Option STR – 2006-2007

287

## Processeur

Transition - stratégies de mise en sommeil du processeur (2/3)

- ❑ Schéma général des stratégies de transition
  - Passage en mode "économe" (ex: sommeil) quand on prédit une période d'inactivité du processeur
  - Exemple de mise en œuvre : quand aucun processus n'est prêt à s'exécuter (boucle idle), passage en mode "économe"
- ❑ Utilisation du mode "tension/fréquence réduit"
  - Description du mode ( $V$  = tension,  $f$  = fréquence)
    - Consommation d'un circuit CMOS : proportionnel à  $V^2f$
    - Consommation par cycle : proportionnel à  $V^2$  : réduction de la consommation de manière quadratique / réduction de tension
    - Réduction de  $V$  doit être accompagnée d'une réduction de  $f$  (réduction de  $V \rightarrow$  "settling time" des portes plus important)
  - Intéressant quand la latence de changement de fréquence  $\ll$  latence de réveil du processeur

Option STR – 2006-2007

288

## Processeur

Transition - stratégies de mise en sommeil du processeur (3/3)

- ❑ Stratégie 1 - (MacOS 7.5). Seuil d'inactivité **fixe**
  - Incrémentation d'un compteur d'activité et d'un compteur d'IO
  - Mise en sommeil du processeur lorsque
    - Pas d'activité processeur pendant plus de 2 secondes
    - Pas d'IO depuis 15 secondes
  - Réveil sur prochain événement
- ❑ Stratégie 2 - (Windows 3.1). Seuil d'inactivité **fixe et nul**
  - Présence d'une boucle IDLE système qui est exécutée dès lors qu'aucune tâche ne peut s'exécuter et lorsqu'aucune interruption n'est à traiter
  - Mise en sommeil immédiate du processeur
  - Réveil sur prochain événement
- ❑ Pour une stratégie optimale, il faut connaître l'avenir

Option STR – 2006-2007

289

## Processeur

Load-change - réduire la consommation du processeur en travaillant sur les traitements

- ❑ Travail sur le code des logiciels exécutés
  - OS: analyse énergétique d'un OS et optimisation des parties gourmandes
  - Code applicatif : utilisation d'instructions moins gourmandes en énergie
  - Utilisation de compilateurs "energy-aware" [TMWL96]

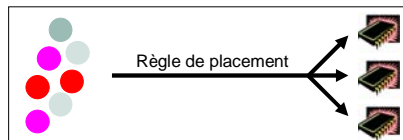
Option STR – 2006-2007

290

## Processeur

Load-change – Distribution et multiprocesseur hétérogène (1/2)

- ❑ Architecture
  - Plusieurs cœurs de processeur différents
  - Tension d'alimentation plus basse que sur un mono-processeur
  - Gain important d'énergie avec potentiel de performance identique
- ❑ Système d'exploitation
  - Distribution des traitements sur l'ensemble des processeurs



- La règle de placement réalise le placement et l'ordonnement

Option STR – 2006-2007

291

## Processeur

Load-change – Distribution et multiprocesseur hétérogène (2/2)

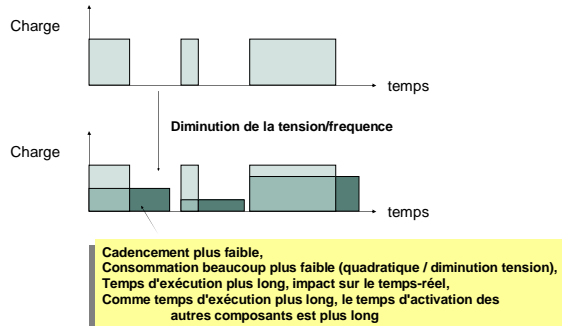
- ❑ Travaux F.Parain (thèse IRISA)
- ❑ Règle de placement (dynamique) pour applications multimédia
  - Temps-réel souple (stratégie au meilleur effort)
    - Placement des traitements en fonction de leurs **échéances** et de leurs **temps d'exécution** prévu (dépend d'un profil d'exécution)
  - Minimisation de la consommation d'énergie relative aux traitements
    - Placement des traitements en fonction de leurs consommations **énergétiques**
  - Combinaison des règles : compromis à trouver
    - Temps réel : efficacité
    - Gestion de l'énergie : consommation

Option STR – 2006-2007

292

## Processeur

Adaptation - changement dynamique de la fréquence du processeur (1/9)



Option STR – 2006-2007

293

## Processeur

Adaptation - Changement dynamique de la fréquence du processeur (2/9)

[WWDS94]

- Division du temps par période de 10-50 ms
- Au début de chaque période, on caractérise l'utilisation du processeur sur la période précédente (adaptatif)
- Choix sur la variation d'utilisation
  - Si l'utilisation croît, on augmente le cadencement du processeur,
  - Si l'utilisation décroît, on diminue le cadencement du processeur
- Evaluation
  - 50% de gain si tension peut être réduite de 5V à 3.3V
  - 70% de gain si tension peut être réduite de 5V à 2.2V
- Difficulté de ce type de méthode : bien prédire l'avenir

Option STR – 2006-2007

294

## Processeur

Adaptation - changement dynamique de la fréquence du processeur (3/9)

- [PBB98] Prédiction plus réalistes en agissant sur une fenêtre de temps plus longue. Différentes variantes :
  - *Past*, prédiction = activité période précédente
  - *Aged*, prédiction = moyenne des activités des périodes précédentes en pondérant plus fortement les périodes les plus récentes
  - *LongShort*, moyenne des 12 activités précédentes. Les 3 dernières ont 3 fois plus de poids
- NB : ces stratégies ne sont pas adaptées dans un contexte temps-réel strict
  - adaptatif
  - risque de rater des échéances si le passé n'est pas représentatif de l'avenir

Option STR – 2006-2007

295

## Processeur

Adaptation - changement dynamique de la fréquence du processeur (4/9)

[PS01] ordonnancement DVS pour temps-réel strict sous EDF

- Rappel ordonnancement EDF
  - Soit un système composé de  $n$  tâches  $T_i$  de période  $P_i$  et de WCET  $C_i$
  - Ordonnancement à priorité dynamique : tâche la plus prioritaire = tâche dont l'échéance absolue est la plus proche
  - Le taux d'utilisation d'un processeur pour un processus est  $U_i = C_i/P_i$
  - La charge processeur totale est
    - $U = \sum_i U_i$
  - Le test de faisabilité pour l'ordonnancement EDF est
    - $U = \sum_i U_i \leq 1$

Option STR – 2006-2007

296

## Processeur

Adaptation - changement dynamique de la fréquence du processeur (5/9)

[PS01] ordonnancement DVS pour temps-réel strict

- Lors la fréquence du processeur est multipliée par un facteur  $\alpha$ , le temps d'exécution d'une tâche  $T_i$  est multiplié par  $1/\alpha$
- Le test de faisabilité d'ordonnancement EDF devient alors pour une fréquence  $\alpha$ 
  - $U = \sum_i U_i \leq \alpha$
  - Possibilité de trouver la meilleure fréquence processeur telles que les échéances sont vérifiées
- Exemple pour un processeur disposant de 3 fréquences de cadencement  $\alpha = (1, 0.75, 0.5)$

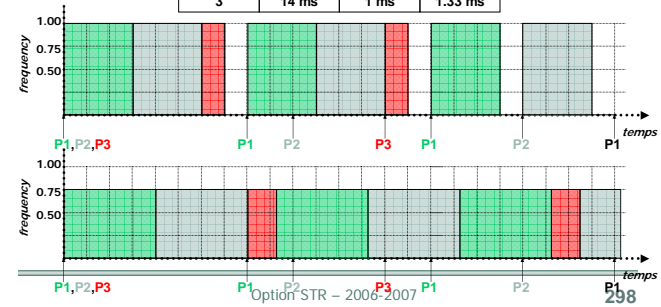
Option STR – 2006-2007

297

## Processeur

Adaptation - changement dynamique de la fréquence du processeur (6/9)

Processus	Période	C $\alpha=1$	C $\alpha=0.75$
1	8 ms	3 ms	4 ms
2	10 ms	3 ms	4 ms
3	14 ms	1 ms	1.33 ms



Option STR – 2006-2007

298

## Processeur

Adaptation - Changement dynamique de la fréquence du processeur (7/9)

- Cycle-conserving RT-DVS (EDF)
  - Prise en compte dynamique du temps d'exécution **effectif** des tâches (plutôt que leur WCET  $C_i$ ) une fois que la tâche s'est exécutée
    - Ajustement **dynamique** de la charge du processeur à partir du temps d'exécution des tâches
  - Quand la tâche  $T_i$  devient prête (début de période)
    - Utilisation  $U_i = C_i/P_i$
  - Quand  $T_i$  se termine **cc, temps d'exécution 'réel'**
    - Ajuster l'utilisation à  $U_i = cc_i/P_i$  pour  $\alpha=1$
  - Comme précédemment, choisir la fréquence pour que  $U = \sum_i U_i \leq \alpha$

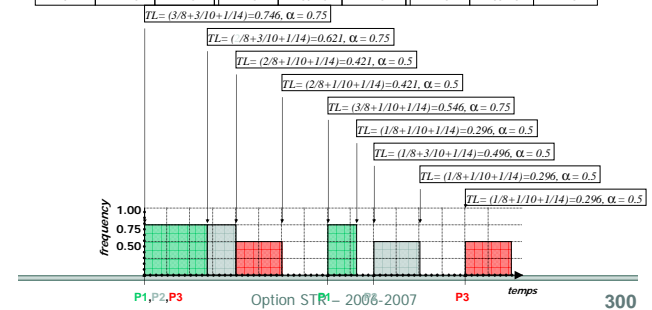
Option STR – 2006-2007

299

## Processeur

Adaptation - changement dynamique de la fréquence du processeur (8/9)

Processus	Période	WCET $\alpha=1$	Instance 1 $\alpha=1$	Instance 1 $\alpha=0.75$	Instance 1 $\alpha=0.5$	Instance 2 $\alpha=1$	Instance 2 $\alpha=0.75$	Instance 2 $\alpha=0.5$
1	8 ms	3 ms	2 ms	2.67 ms	4 ms	1 ms	1.33 ms	2 ms
2	10 ms	3 ms	1 ms	1.33 ms	2 ms	1 ms	1.33 ms	2 ms
3	14 ms	1 ms	1 ms	1.33 ms	2 ms	1 ms	1.33 ms	2 ms



Option STR – 2006-2007

300

## Processeur

Adaptation - changement dynamique de la fréquence du processeur (9/9)

### ❑ Evaluations

DVS method	Energy saved
EDF	0 %
EDF DVS	36 %
EDF DVS conserving	48 %

### ❑ Remarques

- La méthode avec changement **dynamique** de la fréquence est compatible avec une validation **hors-ligne** des contraintes de temps
  - on ne prend en compte les temps effectifs qu'APRES exécution de la tâche
- La méthode avec changement dynamique de la fréquence suppose des temps de changement de fréquence nuls (**pas le cas en pratique**)

Option STR – 2006-2007

301

## Plan

### ❑ Introduction

#### ❑ Les stratégies du système d'exploitation

- concernant le support de stockage
- concernant l'écran
- concernant le processeur
- concernant les périphériques de communication sans fil

### ❑ Conclusion

Option STR – 2006-2007

302

## Périphérique wireless

Caractéristiques (1/4)

### ❑ Cinq modes de fonctionnement typiques

- Transmission
- Réception
- IDLE (consomme toujours de l'énergie)
- Sommeil (consomme très peu d'énergie pour la surveillance des connexions)
- OFF

### ❑ Consommation entre mode réception et IDLE quasiment identique

- ### ❑ Temps de mise en sommeil et de réveil : très variables
- HP HSDL Infrared: 10 µsec mise en sommeil, 40 µsec réveil
  - AT&T WaveLan infrared: 100 ms réveil
  - Metricom Wireless Modem: 5 secondes réveil

Option STR – 2006-2007

303

## Périphérique wireless

Caractéristiques (2/4)

### ❑ La consommation dépend de la distance entre émetteur et récepteur

- ARDIS, système pour longue distance: 40W
- Metricom: 1W
- WaveLan PCMCIA card 1.875W (proximité)
- HDSL infrared: 55mW (proximité)

Option STR – 2006-2007

304

## Périphérique wireless

Caractéristiques (3/4)

### ❑ Bluetooth

- initiée en 1994 par Ericsson
- Technologie de moyen débit 720 kbits/s (1 Mbits/s en débit théorique) à basse consommation énergétique.
- Rayon d'action est limité entre 10 et 30 mètres et ses composants de petite taille lui permettent d'être inséré dans des équipements très mobiles (montre, PDA, voiture, espaces publics...).
- Bluetooth permet de créer un réseau de 8 appareils en communication simultanée.
- Une norme Bluetooth 2 est en cours de spécification : elle devrait permettre d'atteindre un débit théorique de 2 à 10 Mbits/s.

Option STR – 2006-2007

305

## Périphérique wireless

Caractéristiques (4/4)

### ❑ IEEE 802.11b

- Débit théorique de 11 Mbits/s
- Rayon d'action est de 50 à 100 m.
- Avec une antenne amplifiée de 100 milliWatt placée en extérieur, on peut couvrir de 400 mètres à 3 kilomètres
- Utilise la bande de fréquence 2,4 Ghz
- L'IEEE, qui est à l'origine de la norme, travaille à faire évoluer le standard 802.11b pour en améliorer le niveau de sécurité, le débit, la portée et la consommation énergétique.

Option STR – 2006-2007

306

## Périphérique wireless

Transition - mise en sommeil

### ❑ Techniques pour la mise en sommeil identiques à celles mise en place pour les disques

- Néanmoins, le seuil doit prendre en compte les temps nécessaire à la mise en sommeil et au réveil du périphérique (qui peut être très long)
- [SK97] WaveLan Pcmcia pour rappatriement de documents Web via http diminue de 67% la consommation sans perception de latence pour l'utilisateur

Option STR – 2006-2007

307

## Périphérique wireless

Adaptation : extension du protocole

### ❑ Adaptation de la puissance d'émission du périphérique à la distance du récepteur

- Exemple: Réseaux Adhoc / Spontaneous Information Systems
- Problème: estimation de la distance

Option STR – 2006-2007

308

## Périphérique wireless

Load - change - diminution des paquets à émettre

- Extension des fonctionnalités du protocole afin de diminuer le nombre de paquets à émettre
  - Compression de données
  - Suppression de certaines émissions en fonction de la qualité de la transmission
    - Exemple : inutile d'émettre des paquets s'ils vont être perdus
  - Adaptation du contenu à transmettre en fonction des besoins des utilisateurs et des terminaux
    - Exemple : émission d'une image noir et blanc sur un terminal noir et blanc, et de la version couleur sur le terminal couleur
    - Formats "incrémentaux" de données

Option STR - 2006-2007

309

## Conclusion

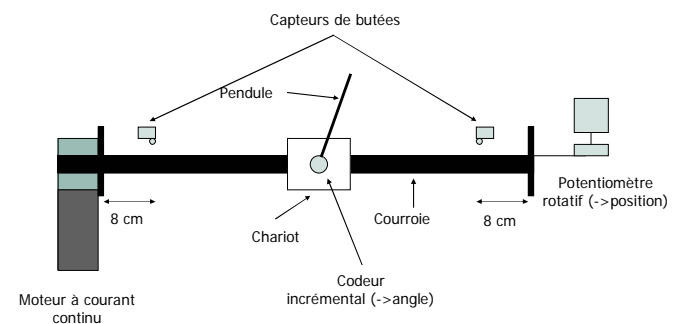
- De nombreux travaux dans un contexte précis d'application
  - Standard (sans contrainte de temps-réel)
  - Temps-réel strict/souple
- Problèmes restant à résoudre
  - Obtention d'une estimation précise de la consommation énergétique est une tâche (exécution en boucle et vidage batterie ?)
  - Moyen de lever ce problème : utilisation de compteurs matériels

Option STR - 2006-2007

310

## Etude de cas Contrôle d'un pendule inversé

## L'architecture

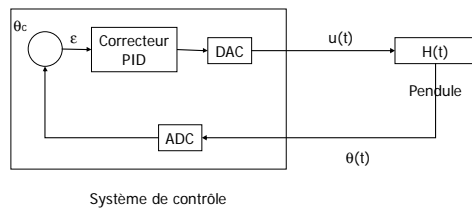


Option STR - 2006-2007

312

## Contrôle du pendule

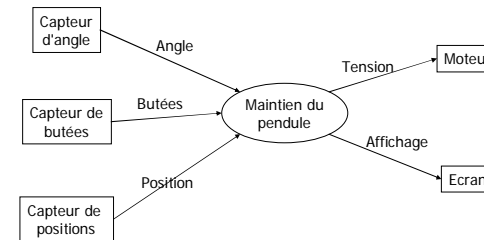
- ❑ Objectif
  - Garder le pendule en position verticale inversée
- ❑ Diagramme fonctionnel initial : circuit en boucle fermée



Option STR – 2006-2007

313

## Diagramme de contexte de l'application



Option STR – 2006-2007

314

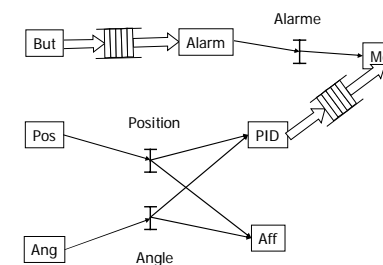
## Description fonctionnelle détaillée de l'application (1/2)

- ❑ Tâches
  - Contrôle butées (But) : lecture capteurs de butées. Déclenche la tâche alarme en lui envoyant si oui ou non on est en butée
  - Alarme (Alarm) : Positionne une variable d'alarme lue par la tâche moteur
  - Contrôle de position (Pos) : lecture et stockage de la position courante
  - Contrôle de l'angle (Ang) : lecture et stockage de l'angle
  - Affichage (Aff) : Affichage graphique de la position et l'angle du pendule
  - PID : asservissement et génération de la tension d'alimentation du moteur (déclenche la tâche moteur)
  - Commande du moteur (Mot)
- ❑ Donner le schéma fonctionnel de l'application en utilisant les notations du cours

Option STR – 2006-2007

315

## Description fonctionnelle détaillée de l'application (2/2)



NB : plusieurs schémas sont possibles

Option STR – 2006-2007

316

### Détermination des paramètres temporels (1/3)

- ❑ Choix : tâches périodiques synchrones
  - Simplicité de mise en œuvre sur les exécutifs temps-réel
- ❑ Contraintes : liées aux caractéristiques physiques du procédé
  - Arrêt du moteur quand un détecteur de butée le signale, et ce avant qu'il n'atteigne la butée
    - Vitesse maximale du chariot : 3 m/s
    - Distance entre capteur de butée et butée : 8 cm
    - Temps de réaction max =  $0.08 / 6 = 26$  ms
    - Sécurité → choix d'une échéance de 25 ms
  - Réaction du système suffisamment rapide pour éviter la chute du pendule
    - Etude de la cinématique du pendule (non détaillée)
    - Délai de réaction = 10 ms
  - Régularité d'exécution de la tâche de contrôle du moteur

Option STR – 2006-2007

317

### Détermination des paramètres temporels (2/3)

- ❑ Dédire des contraintes physiques les relations sur les périodes des tâches
- ❑ Donner des périodes satisfaisant ces contraintes

Option STR – 2006-2007

318

### Détermination des paramètres temporels (3/3)

- ❑ Paramètres choisis (en dixième de ms). Les  $C_i$  sont obtenus par test ou analyse du code

Tâche	ri	Ci	Di	Pi
Ang	0	3	20	20
PID	0	1	10	10
Mot	0	1	10	10
Pos	0	2	20	20
But	0	1	70	70
Alarme	0	1	70	70

Option STR – 2006-2007

319

### Ordonnançabilité et exécution

- ❑ Exécution
  - Coder ce système de contrôle sous VxWorks
- ❑ Ordonnançabilité
  - Assignation de priorités dans l'ordre de déclaration des tâches
    - Le système est il ordonnançable ?
    - Donner un schéma temporel de son exécution
    - Est-ce que la tâche moteur a une gigue de régularité d'exécution ?
    - Le problème d'inversion de priorités se pose t'il ?
  - Assignation de priorités selon RM
    - Mêmes questions
  - Proposer une méthode pour éliminer la gigue

Option STR – 2006-2007

320