Master informatique – première année

Option STR
Systèmes temps-réel

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

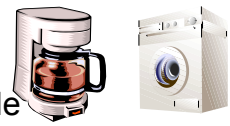
Systèmes temps-réel embarqués
Définition

- 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

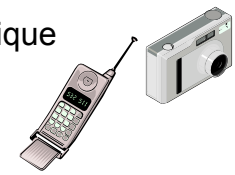
Caractéristiques générales

- 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)

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

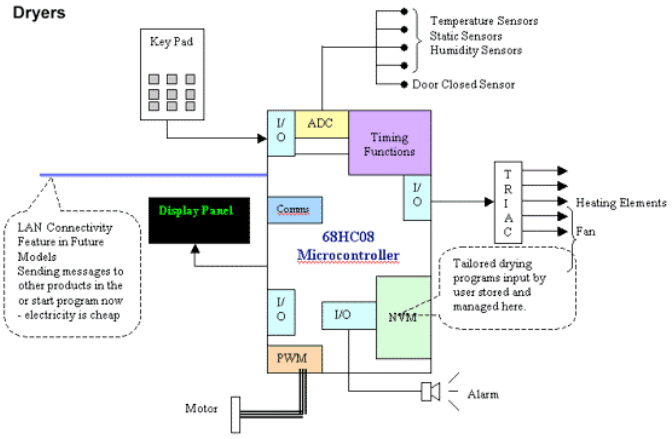


- Automobile
  - Systèmes anti-blocage de freins, contrôle moteur, informatique de confort

- Contrôle de procédés industriels
- Avionique, spatial, production d'énergie (nucléaire)
- Périphériques informatique : FAX, imprimantes

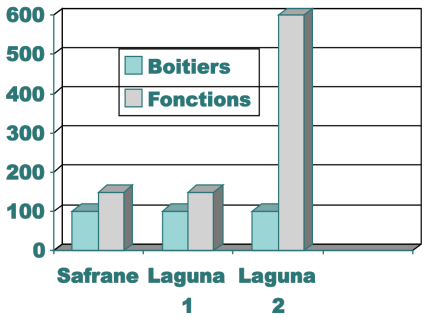
➔ Grande diversité des besoins (puissance, fiabilité, criticité)

Systèmes temps-réel embarqués
Exemple : sêche-linge



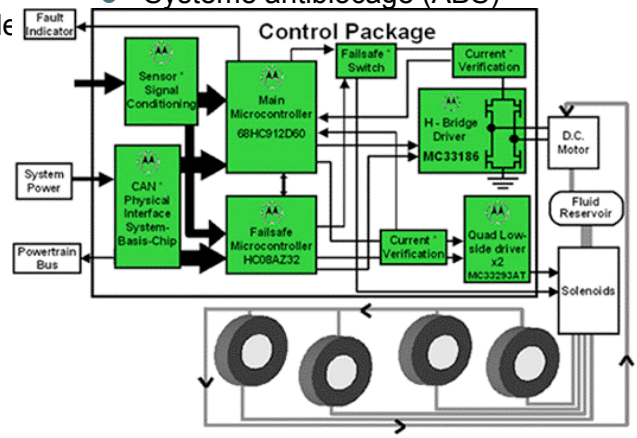
Systèmes temps-réel embarqués
Exemple : automobile

- Evolutions des fonctions logicielles dans l'automobile



(C. Balle, Journée Illiatech, octobre 2001)

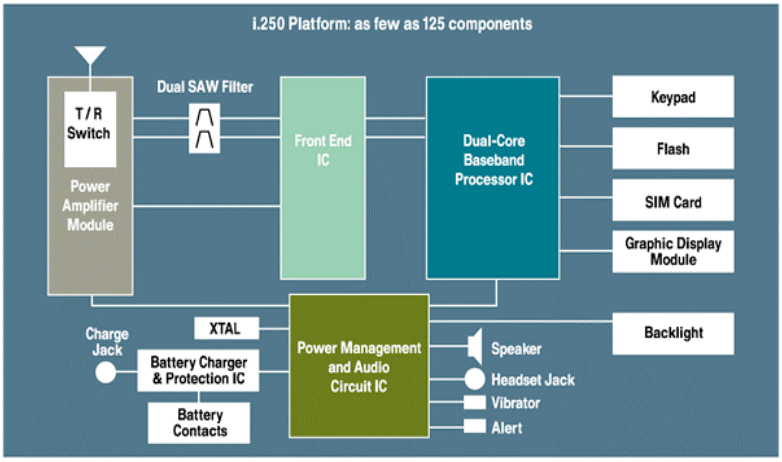
- Système antiblocage (ABS)



(site Web motorola : www.motorola.com)

Exemple : téléphone portable

- Plate-forme I.250 Motorola pour téléphones cellulaires



(site Web motorola : www.motorola.com)

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 »)

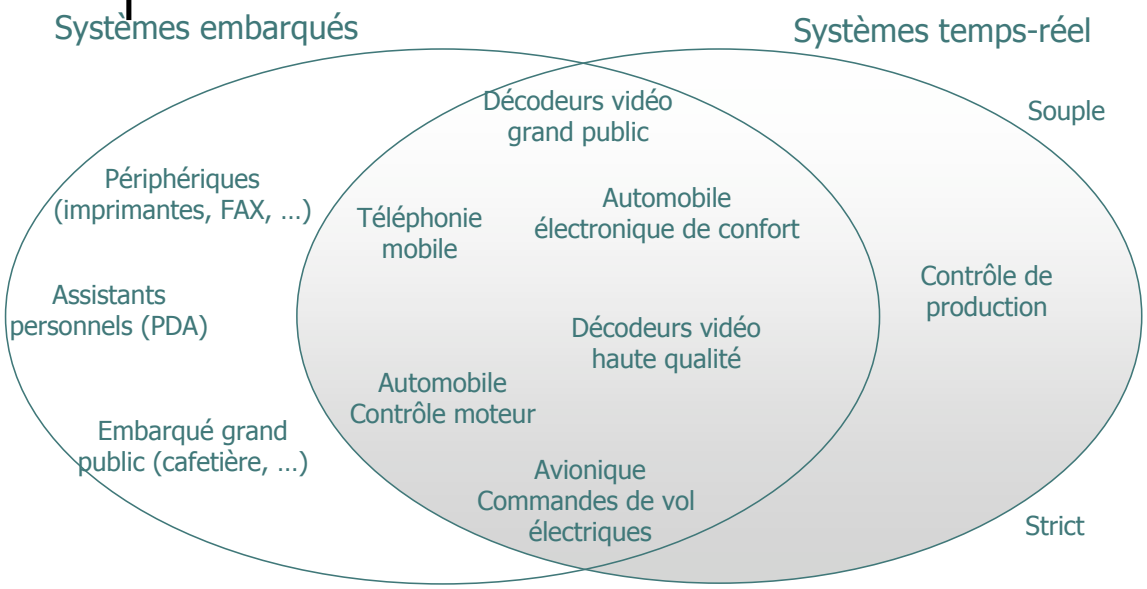
Contraintes de conception (1/2)

- Temps-réel
  - Le temps réel (physique) entre dans le critère de correction des applications
  - Besoin de garanties sur le respect de contraintes temporelles (prévisibilité)
- Sûreté de fonctionnement
  - Définition : capacité à placer une confiance justifiée dans le service
  - Entraves : défaillances provenant de fautes
  - Moyens pour la sûreté 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

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

Temps-réel vs embarqué



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'ordonnançabilité et obtention de pires temps d'exécution
  - Ordonnancement hybride de tâches temps réel strict et non strict
- Minimisation de la consommation énergétique

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

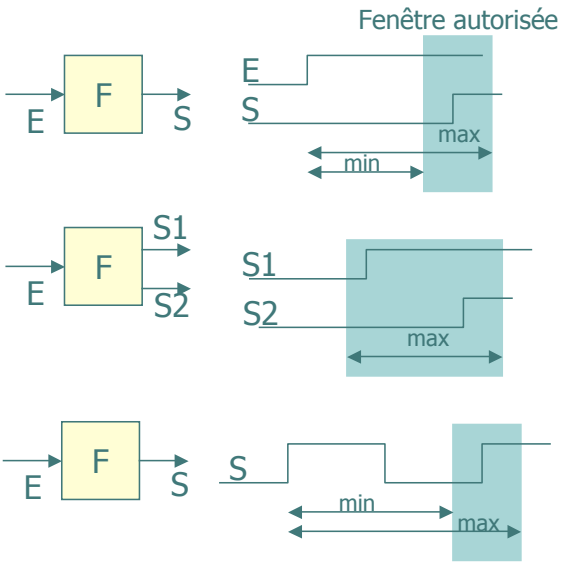
Notion de temps-réel

- 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)
- 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

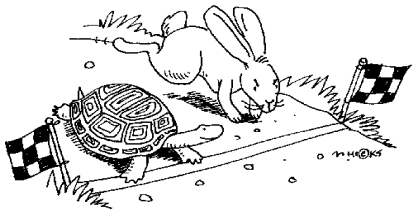


Classes de systèmes temps-réel

- 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'avionique, 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'usager
  - Pas temps-réel tel qu'on l'entend dans ce cours

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 ¨

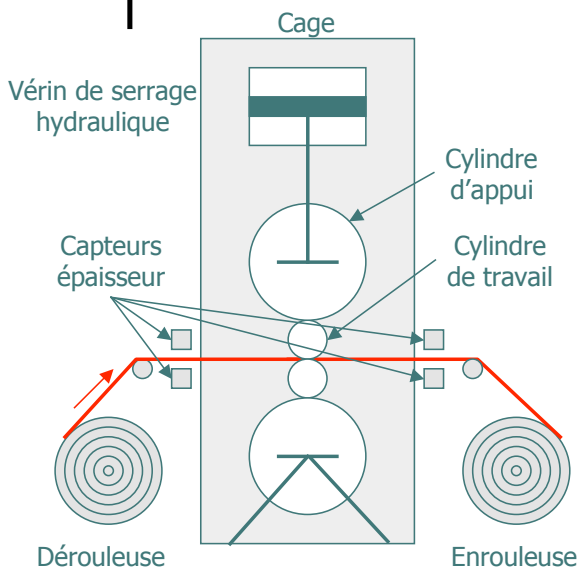


  - Il existe des contraintes strictes qui ne sont pas courtes : dépend de l'environnement
  - La puissance de calcul : condition nécessaire mais pas suffisante
  - Vitesse moyenne ou vitesse maximale n'est pas importante, le pire-cas importe
    - Murphy's law : if something can go wrong, it will go wrong

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
  - Ordonnancement des calculs. Pas « à la main » : méthodes d'ordonnancement 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
  - Conditions (analyse) d'ordonnançabilité

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

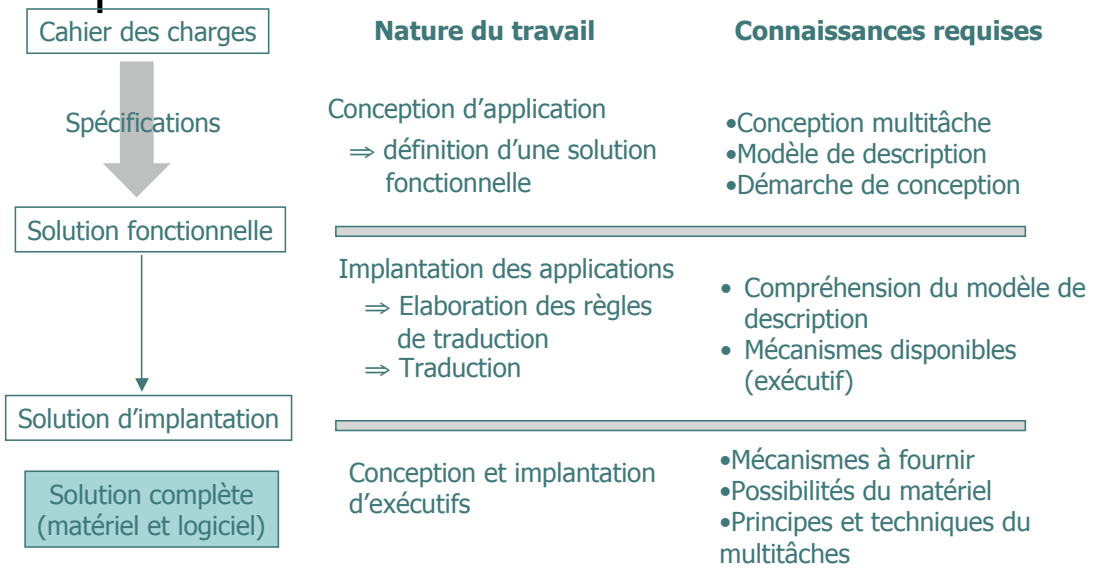


- Laminage à froid d'aliminium
- Lubrification au Kérosène
- Epaisseur sortie : 0.25 à 0.45 mm
- Vitesse bande : 108 km/h
- Systèmes asservis pour :
  - Régulation épaisseur de sortie
  - Régulation de planéité
  - Régulation vitesses moteurs, …
- Contraintes spécifiques
  - Disponibilité : 24h/24h, arrêt max pour maintenance de 16h/semaine
  - Sécurité : rupture de bande, feu dans kérosène

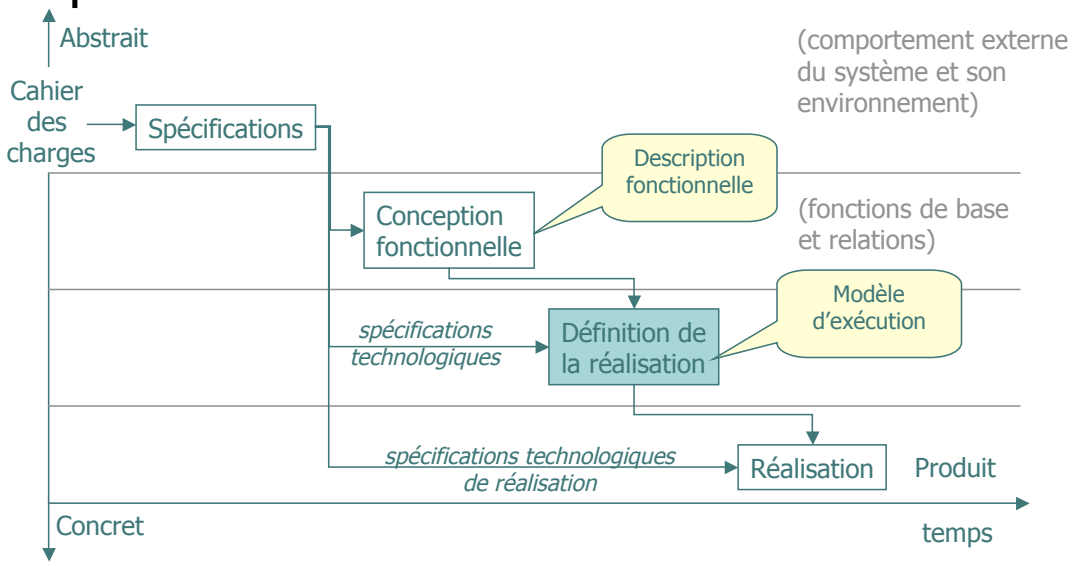
Exemple d'application temps-réel (2/2)
Le laminoir [Cottet, 2000]

- 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

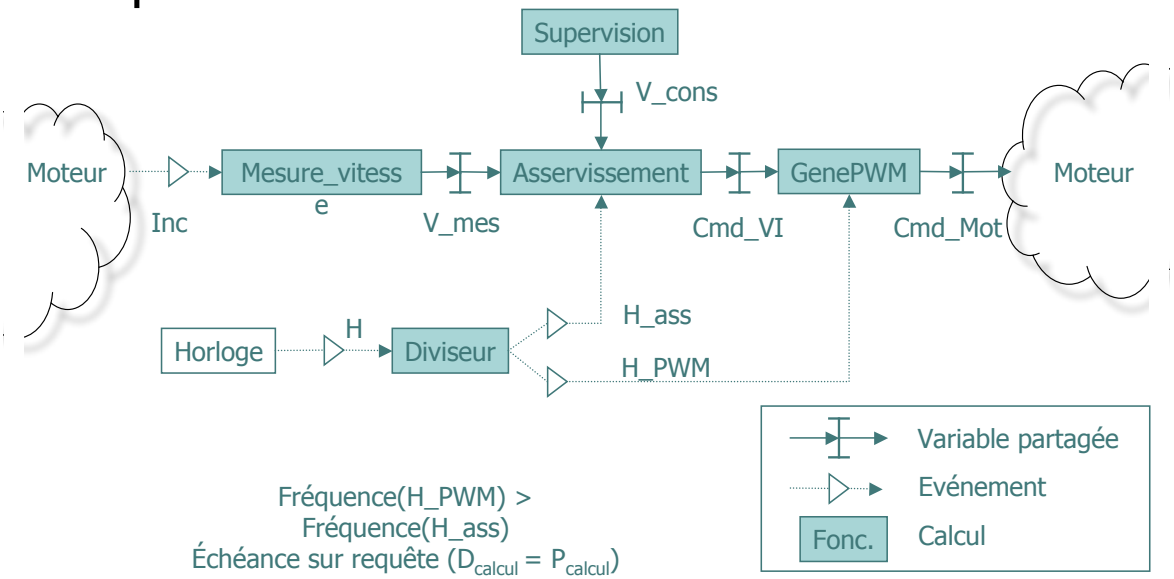
Niveaux d'approche pour la conception d'applications (temps-réel)



Situation dans le processus de développement (1/2)



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



Fréquence(H_PWM) > Fréquence(H_ass)
Échéance sur requête ($D_{calcul} = P_{calcul}$)

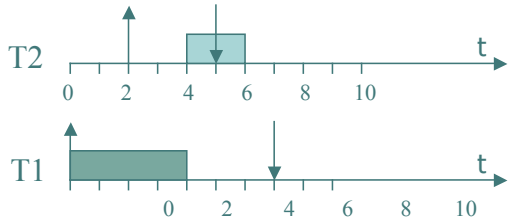
Ordonnancement temps-réel

Introduction (1/2)

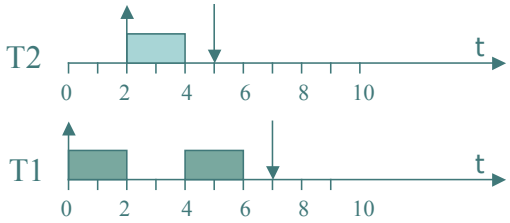
- Définition
  - Ensemble des règles définissant l'ordre d'exécution des calculs sur le processeur
- Pourquoi ordonnancer ?
  - Parce que ça a un impact sur le respect des contraintes de temps
  - Exemple :
    - Tâche T1 : arrivée en 0, temps d'exécution 4, échéance 7
    - Tâche T2 : arrivée en 2, 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

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

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)

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

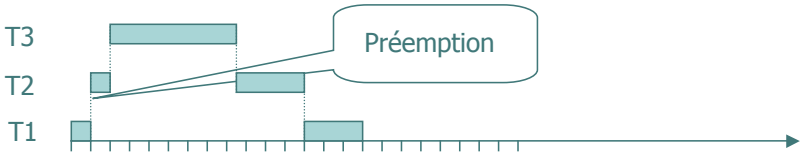
Classification (3/5)
Ordonnancement non préemptif / préeemptif

- 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

Classification (4/5)

- Ordonnancement conduits par la priorité
  - Ordonnancements préemptifs
  - C'est la tâche de plus haute priorité qui s'exécute
- Priorités fixes (statiques) / priorités dynamiques
  - Fixes : indépendants du temps, fixées a priori
  - Dynamiques : évoluent avec le temps
- Exemple
  - prio(T3) > prio(T2) > prio(T1) (ici, priorités fixes)
  - T1, T2 et T3 arrivent respectivement aux dates 1, 2, et 3

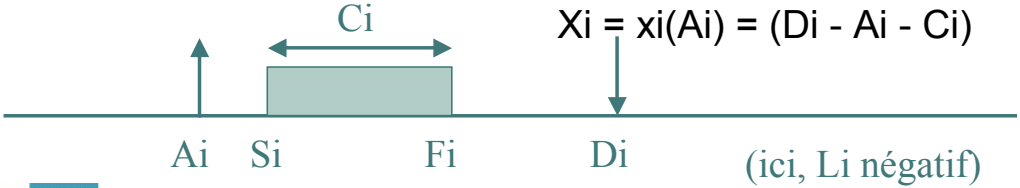


Classification (5/5)

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

Notations (1/2)

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



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
    - Hyperpériode: cycle d'ordonnancement: intervalle [0,PPCM(Pi)] pour tâches synchrones, [min(Oi), max(Oi,Oj+Dj) + 2 * PPCM(Pi)] sinon
  - 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

Quelques ordonnancements

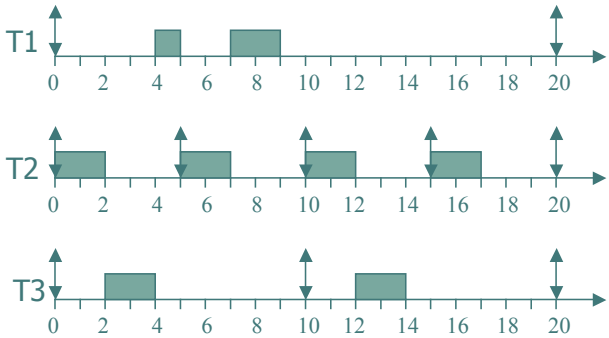
- Ordonnancements non préemptifs
  - ***Selon l'ordre d'arrivée : premier arrivé – premier servi (FIFO)***
  - ***Selon la durée de calcul : plus court d'abord***
  - Exploration des ordres possibles [Bratley, 1971, Spring, 1987]
    - Exploration exhaustive (branch-and-bound) : complexité algorithmique (problème NP-complet)
    - Recherche orientées ou heuristiques pour réduire la complexité
- Ordonnancements 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

Ordonnancement 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 (Di=Pi).

Ordonnancement préemptif à priorité fixe
Rate Monotonic (2/2)

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



Prio(T2) > Prio(T3) > Prio(T1)
Exécution cyclique
(PPCM des périodes)

Ordonnancement
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 : $U = \Sigma C_i/P_i$
- Remarque : Pour que le 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

Ordonnancement préemptif à priorité fixe
Deadline Monotonic (1/2)

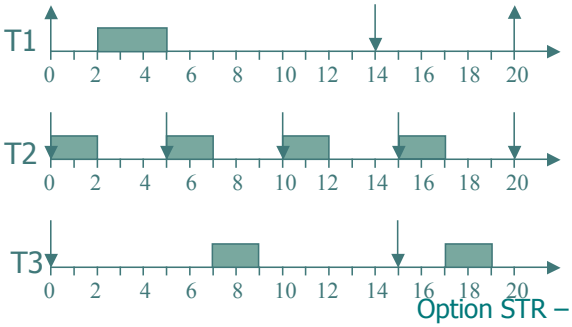
- Pour tâches périodiques
- Définition [Leung & Whitehead, 1985]
  - La priorité d'une tâche est inversement proportionnelle à son échéance relative (conflits résolus arbitrairement)
- Propriété
  - Optimal dans la classe des algorithmes à priorités fixes pour des tâches périodiques indépendantes à échéance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

Ordonnancement préemptif à priorité fixe
Deadline Monotonic (2/2)

- Exemple : ordonnancement 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)



Ordonnancement préemptif à priorité dynamique
Earliest Deadline First (1/2)

- Earliest Deadline First (EDF)
- Applicable pour tâches périodiques et non périodiques

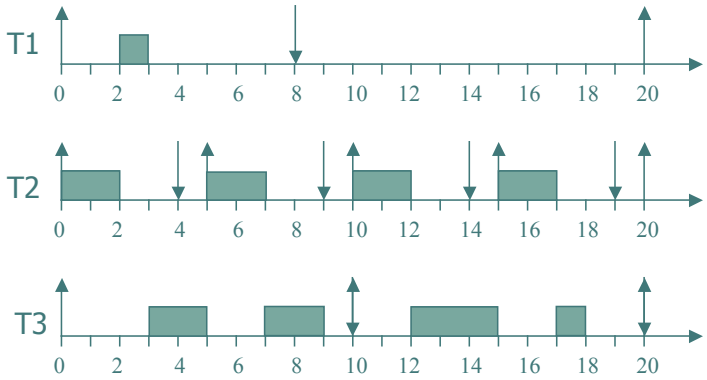
- Définition [Liu & Layland, 1973]
  - A un instant donné, la tâche la plus prioritaire (parmi les tâches prêtes) est celle dont l'échéance absolue est la plus proche
  - Préemptif

- Propriété
  - Optimal dans la classe des algorithmes préemptifs pour des configurations de tâches périodiques indépendantes avec échéance ≤ période

Ordonnancement préemptif à priorité dynamique
Earliest Deadline First (2/2)

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

(2 préemptions en 5 et 15)



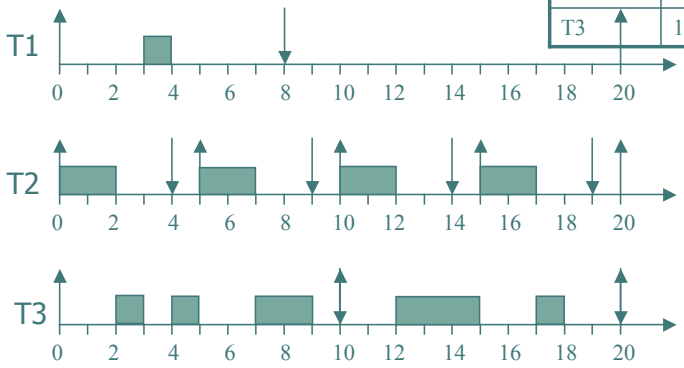
Ordonnancement préemptif à priorité dynamique
Least Laxity First (LLF) (1/2)

- 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 xi(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 ≤ période

Ordonnancement 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



(3 préemptions en 3, 5 et 15)

(Ici, en cas de laxité égale, sélection tache de plus petit numéro en t=3)

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 EDF que pour RM/DM

$$\sum_{i=1}^{n} \frac{Ci}{Pi} \leq n(2^{\frac{1}{n}}-1) \qquad \sum_{i=1}^{n} \frac{Ci}{Pi} \leq 1$$

  - Mais mauvaise tolérance aux surcharges d'EDF (effet domino)
- Mise en œuvre
  - Ordonnancement hors-ligne: simple (pas besoin d'un OS)
  - Ordonnancement en-ligne: priorités statiques plus faciles à mettre en œuvre

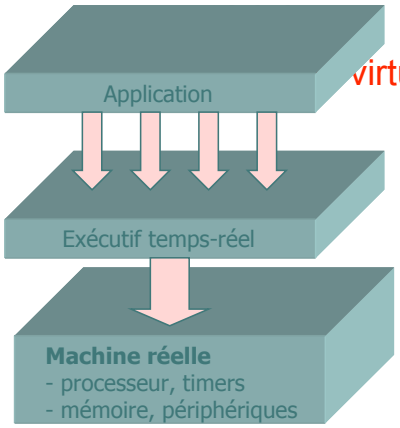
Exécutifs temps-réel multitâches

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 ⇒ partage du processeur, ordonnancement des fonctions
    - Certains jeux de tâches respectent leurs échéances uniquement en préemptif : cf exemple introductif

Rôle d'un exécutif temps-réel

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



Appels systèmes (primitives)

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

Gestion du matériel

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

⇒ Plus besoin de manipulations du matériel

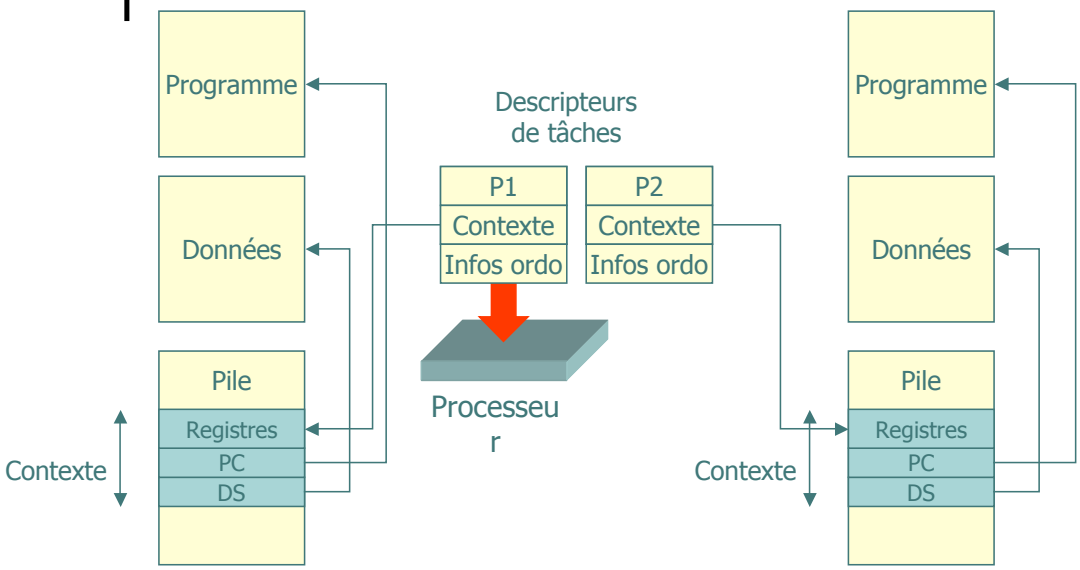
Types de services fournis

- 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

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

Gestion de tâches
Terminologie (2/2)



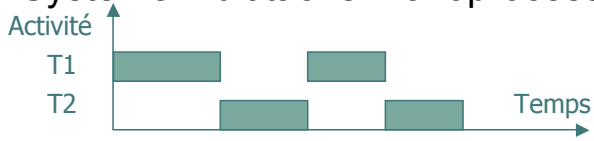
Niveaux de parallélisme (1/2)

- Parallélisme réel
  - Systèmes mutiprocesseurs 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

Niveaux de parallélisme (2/2)

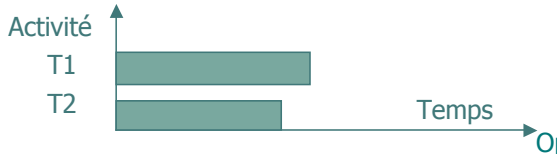
- Système multitâche monoprocesseur non préemptif

- Système multitâche monoprocesseur préemptif

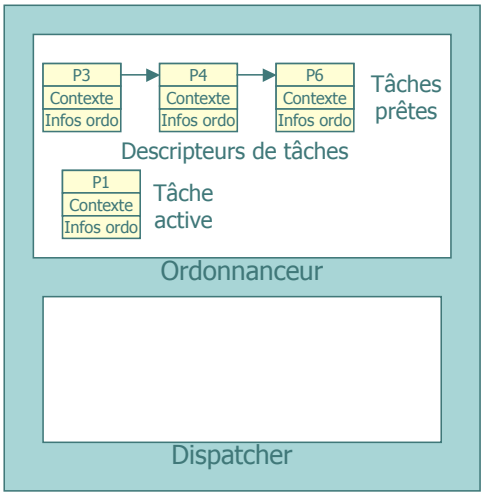


Entrelacement entre T1 et T2 dépend du type d'ordonnancement

- Système multitâche multiprocesseurs



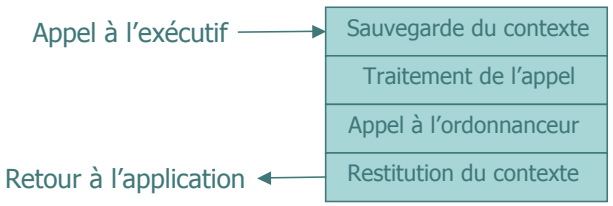
Gestion de tâches
Structure de l'exécutif



- Ordonnanceur : choix de la tâche à exécuter selon une politique d'ordonnancement
- 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

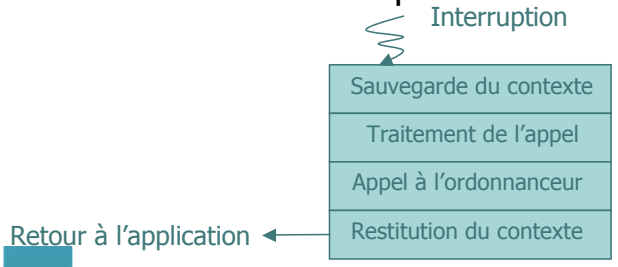
Gestion de tâches
Appels système et interruptions

- Appel système

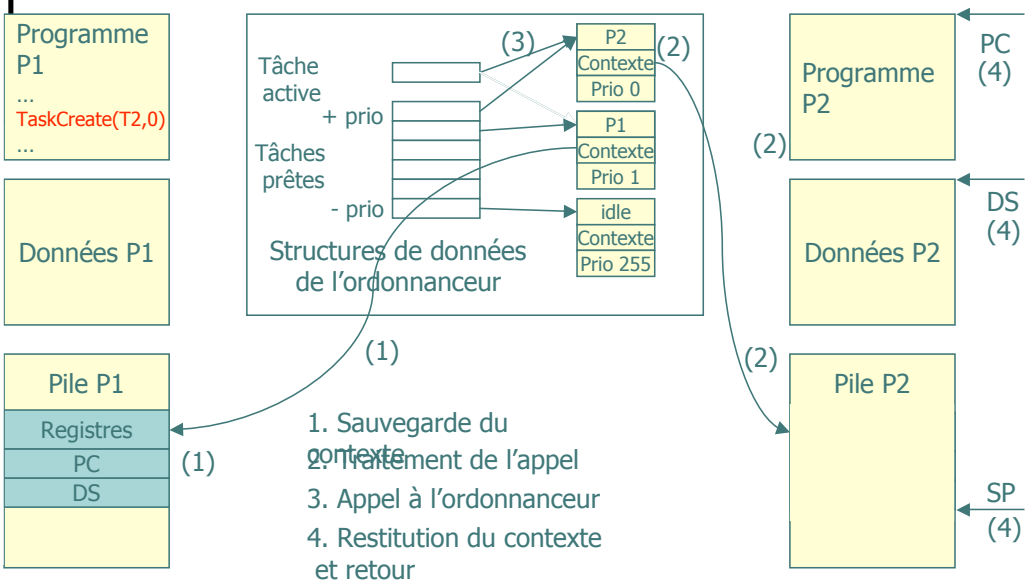


- Gestion d'interruption

On ne retourne pas nécessairement dans la tâche appelante/interrompue

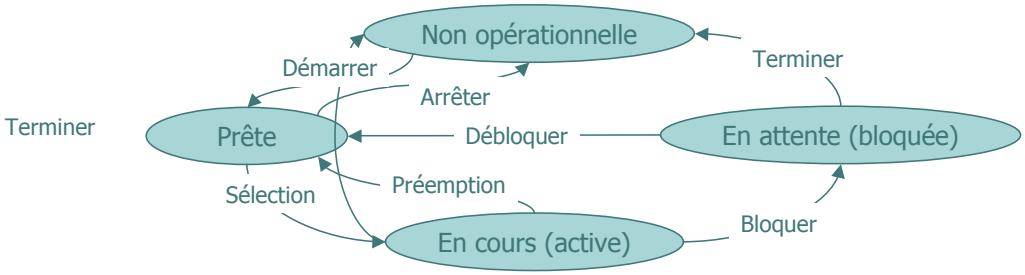


Gestion de tâches
Exemple de déroulement d'un appel système



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



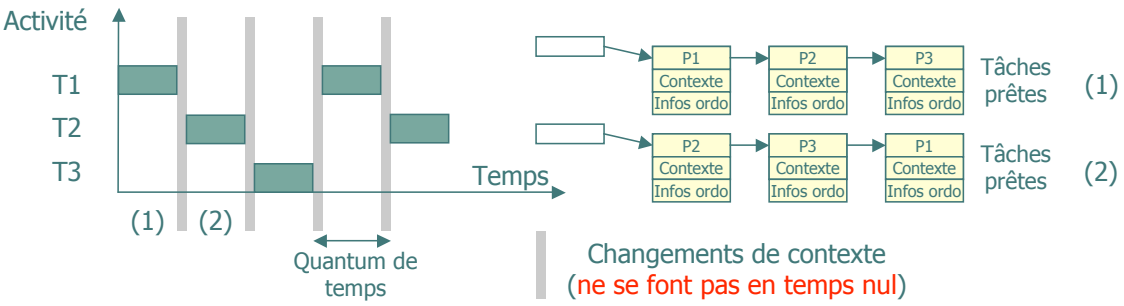
Terminer, Arrêter, Démarrer : primitives (⇒ appelées par les tâches) de gestion de tâches
Bloquer, Débloquer : primitives de synchronisation
Sélection, Préemption : décisions internes de l'ordonnanceur

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

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

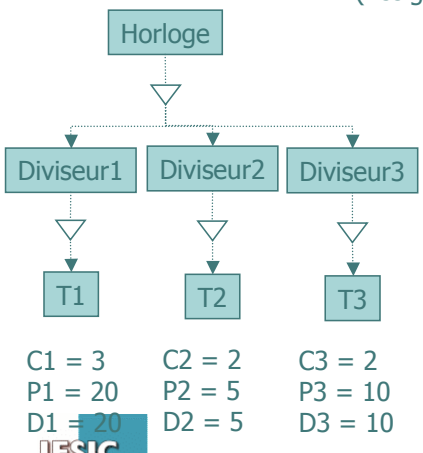
Gestion de tâches
Ordonnanceurs temps-partagé (2/2)

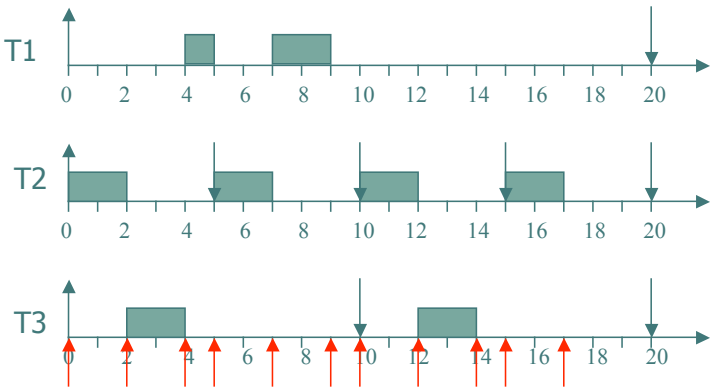
- Limitations
  - Pas de prise en compte des importances relatives des tâches
  - Difficulté de choix de la tranche de temps
    - si trop grand, augmente les temps de réponse
    - si trop petit, trop de surcoût dus aux changements de contexte
  - Peu de résultat pour vérifier a priori le respect des échéances
- 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)

Gestion de tâches
Ordonnanceurs à priorités

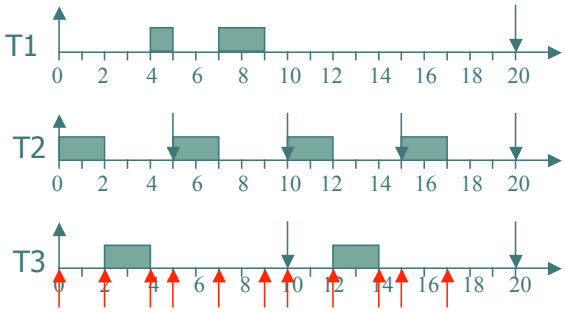
- Assignation d'une priorité à chaque tâche
- Allocation du processeur (état = active) à la tâche prête la plus prioritaire

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





Gestion de tâches
Ordonnanceurs à priorités



Intervention de l'ordonnanceur
(arrivées de tâches, fins de tâches)

0 : T2 T3 T1 (arrivée de T1, T2 et T3)
2 : T3 T1 (fin de T2) ⇒ exec T3
4 : T1 (fin de T3) ⇒ exec T1
5 : T2 T1 (arrivée de T2) ⇒ préemption de T1 par T2
7 : T1 (fin de T2) ⇒ exec T1
9 : vide (fin de T1)
10 : T2 T3 (arrivée de T2 et T3) ⇒ exec T2
12 : T3 (fin de T2) ⇒ exec T3

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'usager
  - Pas de notion de contrainte de temps
  - Pas de vérification des contraintes de temps

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)

Gestion de tâches
Exemples de primitives

Primitive	OSEK/VDX	VxWorks

Démarrage	ActivateTask(tid);
	
	Schedule(); /* *Appel explicite à l'ordonnanceur* */
		taskInit(adtcb,Nname,Prio,Opt,adstack,StSize,adstart, …);
		tid = taskSpawn(Name,Prio,Opt,,StSize,adstart);
		taskActivate(tid);

Arrêt	TerminateTask(); ChainTask(tid); /* *Arrêt au profit de tid* */	taskDelete(Tid);
Manipulation priorités	Aucune (priorités non modifiables)	taskPrioritySet(Tid,priority); taskPriorityGet(Tid,adpriority);
Suspension / reprise	Aucune	taskSuspend(Tid); taskResume(Tid);

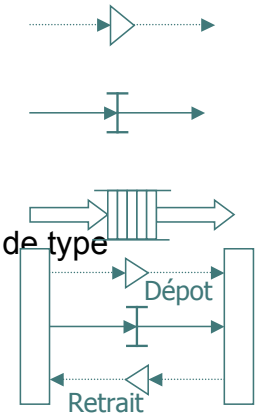
Gestion de tâches
Choix d'une politique d'ordonnancement

- 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 œuvre plus important (n log N pour tri)

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



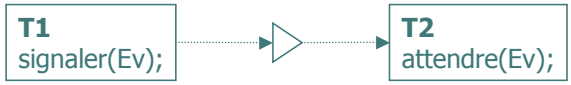
Synchronisation

- 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 : sémaphores

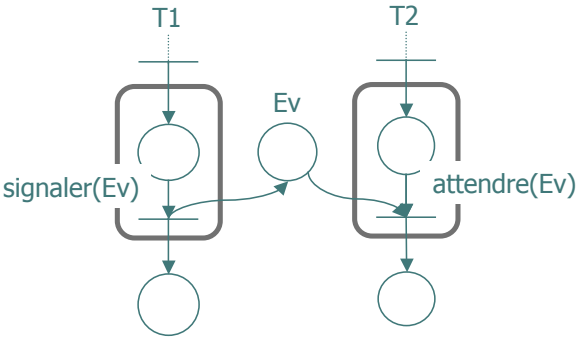
Synchronisation

- 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)

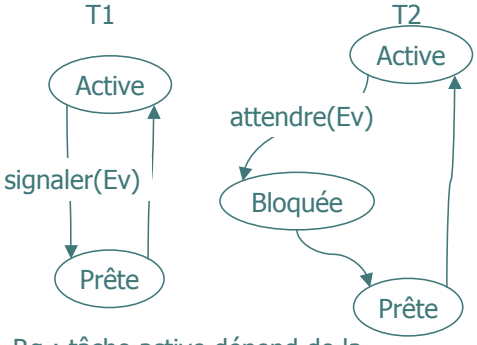
Synchronisation



Description du comportement (réseaux de Petri)

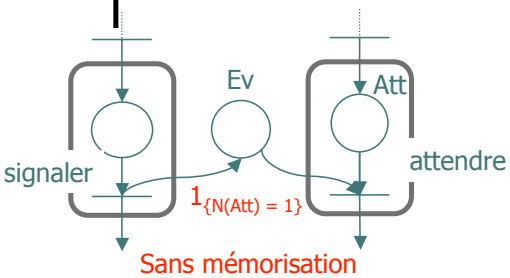


Evolution de l'état des tâches

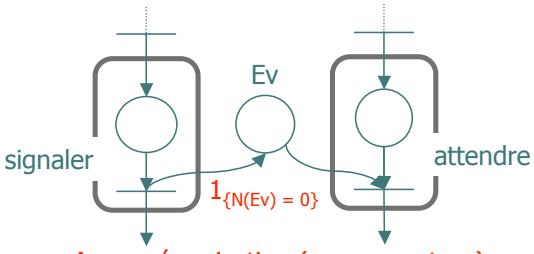


Rq : tâche active dépend de la politique d'ordonnancement

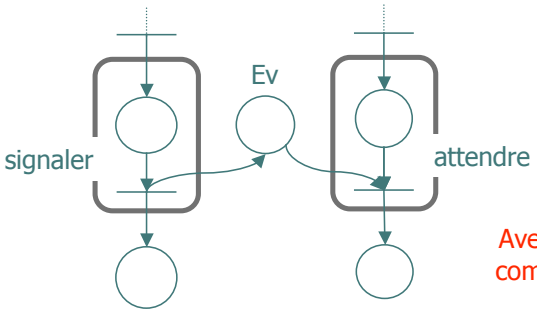
Synchronisation



Sans mémorisation



Avec mémorisation (sans compteur)



Avec mémorisation et compteur (sémaphore)

Synchronisation par événements
Exemple OSEK/VDX

- Types d'événements
  - Evénements avec mémorisation sans compte, effacement explicite Evénements 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)

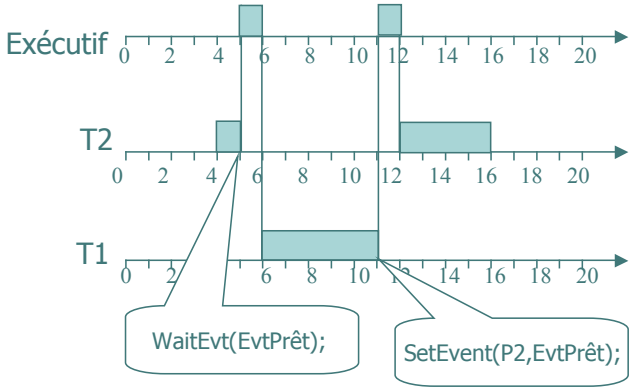
Synchronisation par événements
Exemple OSEK/VDX

- Modèle fonctionnel

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

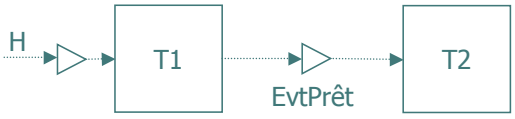
Synchronisation par événements
Exemple OSEK/VDX

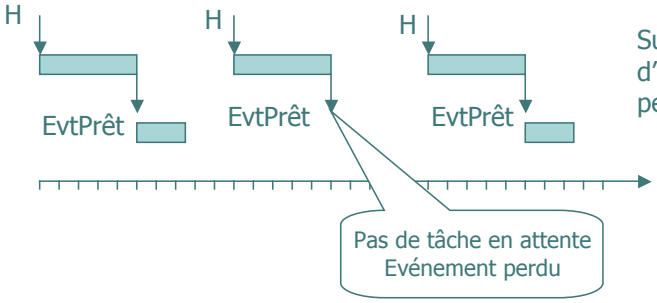
- Chronogramme de l'exemple précédent (T2 plus prioritaire que T1)



Synchronisation par événements
Exemple de pertes d'événements

- Evénements sans mémorisation (pSOS, Sceptre)





Sur cet exemple, pas de perte d'événement si T1 et T2 périodiques avec même période

Synchronisation par événements
Implantation

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

```
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 */
```

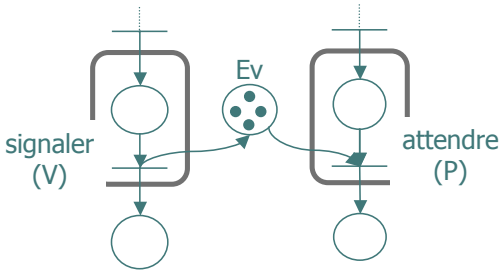
```
Signal(int tid, int e) {
    Sauvegarde contexte;
    Ev[e] = Set;
    if (T[tid].état == Bloqué) {
        T[tid].état = Prêt;
        Ev[e] = Clear;
    }
    Appel à l'ordonnançeur;
    Restitution du contexte;
}
```

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

Pas de compteur un booléen suffit

Synchronisation par sémaphores
Principe

- Primitives classiques
  - sid = create_sema(cpt); /* *cpt >= 0* */
  - P (sid); /* *Puis-je ?* */
  - V (sid); /* *Vas - y* */
- Comportement



Valeur initiale = nombre de jetons initialement dans Ev (nombre de P possibles avant blocage)

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 */
```

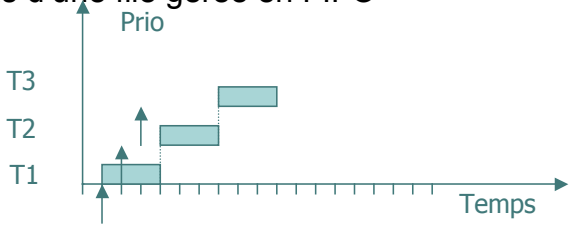
```
V (t_sema *s) {
    Sauvegarde contexte;
    (s->cpt) ++;
    if (s->cpt ≤ 0) {
        T = retirer(s->F);
        T->état = Prêt;
    }
    Appel à l'ordonnançeur;
    Restitution du contexte;
}
```

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


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

Illustration sur la synchronisation par sémaphores

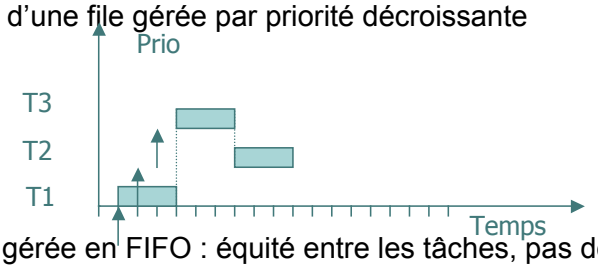
- Code des tâches
  s = create_sema(1);
  T1 : P(s); A1; V(s);
  T2 : P(s); A2; V(s);
  T3 : P(s); A3; V(s);
  Arrivée de T1 en 1, de T2 en 2, de T3 en 3. prio(T1) < prio(T2) < prio(T3); Durée des Ai de 3;
- Cas d'une file gérée en FIFO



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

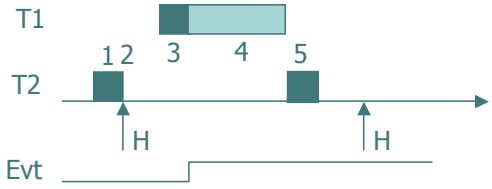
Synchronisation
Besoin d'indivisibilité (1/2)

Exemple mettant en évidence le problème

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

```
T2 {
  cycle
    Wait(Evt);
    Traitement_T2;
  fincycle;
}
```

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




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

H

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

⇒ Exécution INDIVISIBLE des primitives

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)

Synchronisation par événements
Exemples de primitives

Primitive	OSEK/VDX	VxWorks (non exhaustif)
Création / initialisation	Aucune (nb borné d'événements par tâche)	oldhandler = signal (signo, ,handler);
Signal	SetEvent(tid,masque);	kill(tid,signo); raise(signo); /* Auto-signal */
Attente / effacement / consulation	WaitEvent(masque); ClearEvent(masque); GetEvent(tid,&masque);	sigsuspend(masque); sigtimedwait (masque,info,timeout);
Destruction	Aucune	

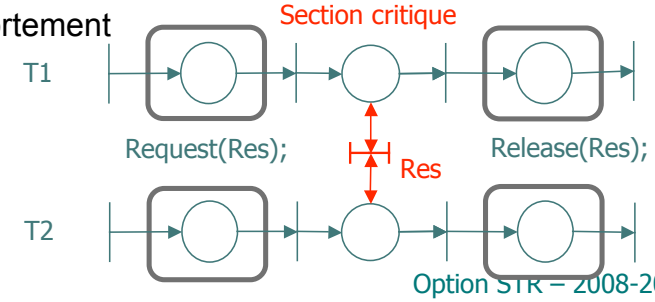
Synchronisation par sémaphores
Exemples de primitives

Primitive	OSEK / VDX	VxWorks
Initialisation	Pas de sémaphore	sid = semCCreate(Opt,Initialcount);
Destruction		semDelete(sid);
Acquisition (« Wait »)		semTake(semid,timeout);
Libération (« signal »)		semGive(semid); semFlush(semid); */* Libère toutes les tâches en attente */*
Consultation		semShow(semid,level);

Partage de ressources

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

- Comportement

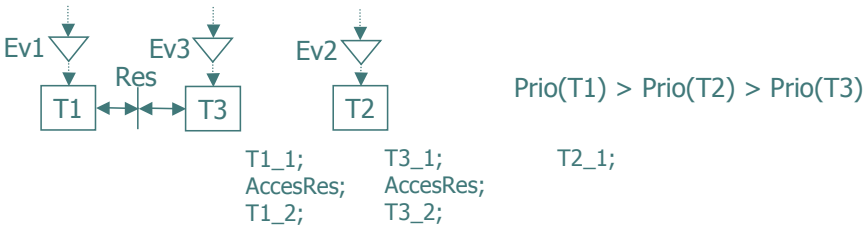


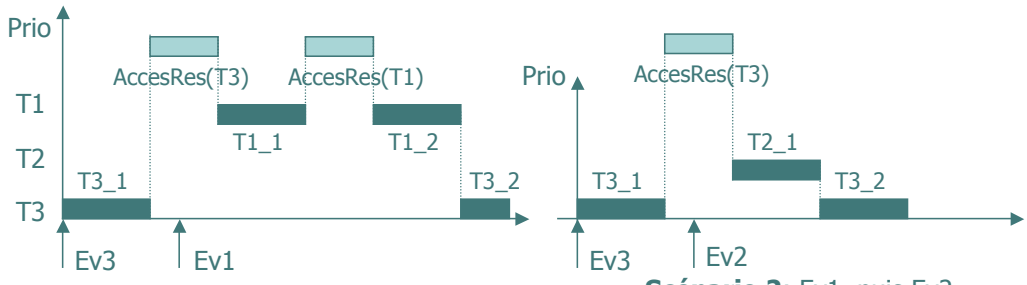
Partage de ressources
Implantation d'une section critique (1/6)

- Supression/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

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





Scénario 1: Ev1, puis Ev3

Scénario 2: Ev1, puis Ev2
A2 bloqué alors que non concerné (et plus prio)

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

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

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

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

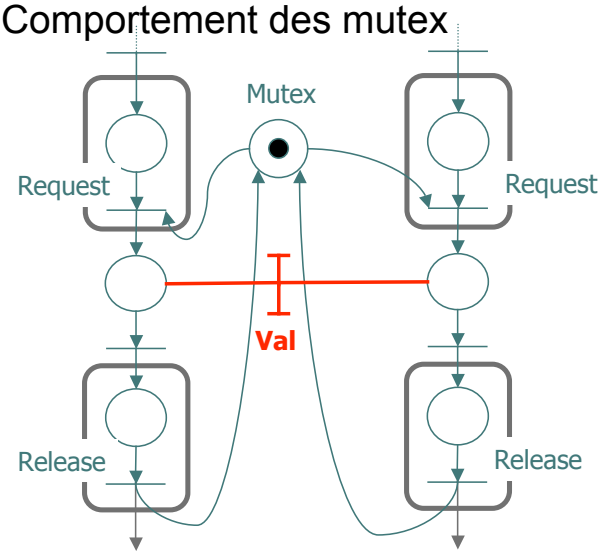
  - Condition de correction
    - Assurer l'indivisibilité
    - mêmes méthodes que pour la synchronisation par événements
  - Problème
    - Monopolisation du processeur
    - Situation de famine quand une tâche prioritaire attend une ressource possédée par une tâche moins prioritaire : ne fonctionne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

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'emboiter dans la même tâche (sans blocage) des accès à la même ressource (ex: VxWorks)

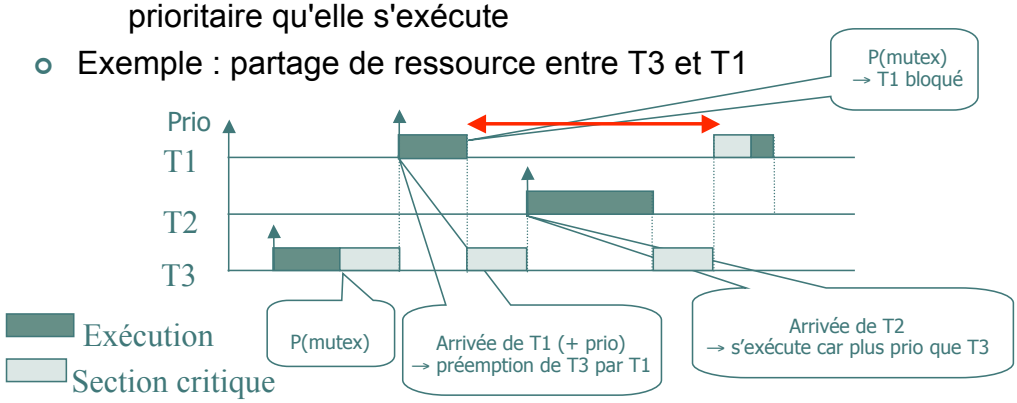
Partage de ressources
Implantation d'une section critique (5/6)

- Comportement des mutex



Partage de ressources
Problème d'inversion de priorité

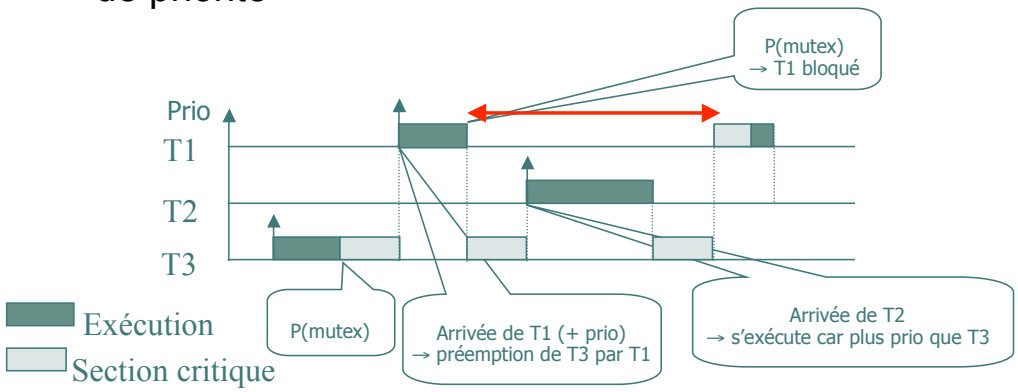
- Définition de l'inversion de priorité
  - une tâche prioritaire est bloquée alors qu'une tâche moins prioritaire qu'elle s'exécute
- Exemple : partage de ressource entre T3 et T1



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

Partage de ressources
Problème d'inversion de priorité

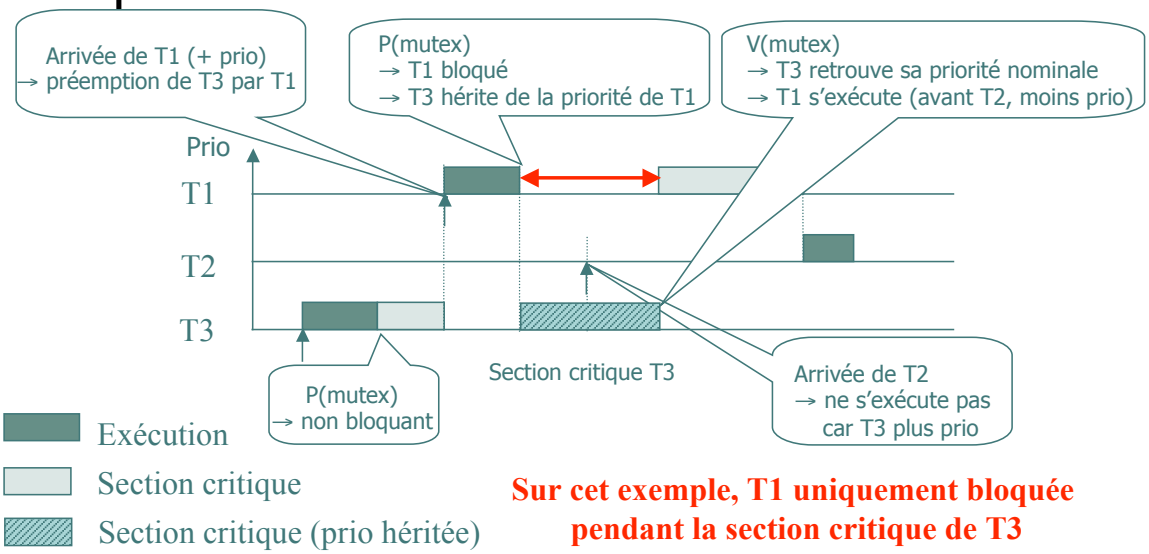
- On voit sur l'exemple précédent qu'une gestion des files d'attente par priorité n'élimine pas l'inversion de priorité



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
  - Ordonnancement à priorité fixe, priorité nominale vs priorité courante
  - Mutex pour sections critiques, section critiques "properly nested"
- Définition du protocole
  - Règle d'ordonnancement: Exécution selon priorité nominale, sauf règle 3
  - Règle d'allocation de ressource
    - Ressource libre : acquisition de la ressource
    - Ressource occupée : blocage
  - Règle d'héritage de priorité : qd blocage d'une tâche T lors de l'accès à R, la tâche bloquante Tl hérite de la priorité courante de T; utilisation par Tl de la priorité héritée jusqu'à libération de R, où elle retrouve la priorité de la tâche la plus prioritaire encore bloquée sur R

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



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

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

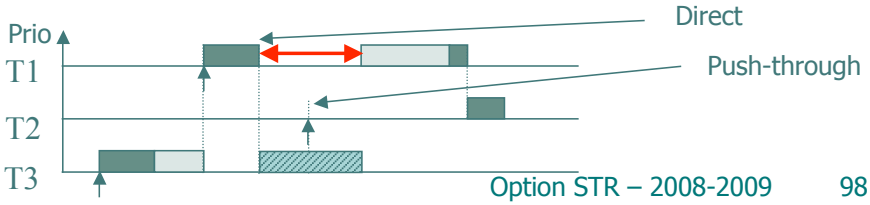
- Propriétés
  - Temps de blocage borné (sauf situation d'interblocage)
    - ➔ Possibilité de vérification a priori des échéances
  - Interblocages possibles
  - Pour implanter PIP, il n'est pas nécessaire de connaître quelle tâche utilise quelle ressource (par contre, nécessaire pour calculer les temps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

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

- Remarque: héritage transitif en cas de sections critiques emboitées
  - 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

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



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

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

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

- Algorithme de calcul du blocage maximal subi par Ti

1. Pour chaque tâche Tj de priorité inférieure à Ti
   - Identifier les sections critiques de Tj pouvant bloquer Ti et prendre la plus longue
   - Sommer les durées des n sections critiques ainsi identifiées pour l'ensemble des Tj
2. Pour chaque sémaphore d'exclusion mutuelle sm
   - Identifier les m sections critiques protégées par sm qui peuvent bloquer Ti 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

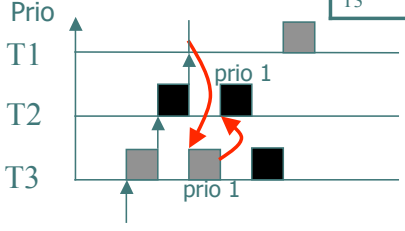
Partage de ressources
Solution à l'inversion de priorité : PIP (8/9)

- Calcul du temps de blocage sur l'exemple (tâche T1)
  - Item 1
    - Tâche T2
      - pas de blocage direct (pas de partage de ress. entre T1 et T2)
      - pas de push-through (ni T2 ni T3 ne peuvent hériter la priorité de tâches plus prioritaires que T1)
    - Tâche T3
      - blocage direct (partage de R avec T3) : n=1
      - pas de push-through blocking
  - Item 2 (examen de tous les sémaphores pouvant bloquer T1, idf m)
    - un seul sémaphore utilisé (pour protéger R). Il peut provoquer uniquement un blocage direct sur T1. m=1
- min(n,m) = la section critique protégée par R

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 Ai	WCET Ci	prio(Ti) (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 T3 nul,
Blocage de T3 à la date 3 non comptabilisé dans son temps de blocage maximal

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

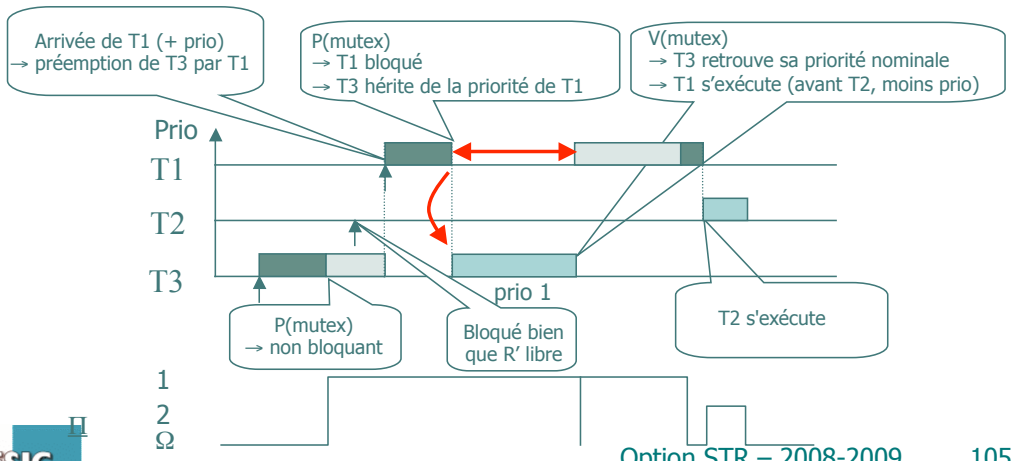
- PCP = Priority Ceiling Protocol (Protocole à priorité « plafond »)
  - ex: OSEK/VDX
- Prérequis
  - Ordonnancement à priorité fixe
  - Ressources utilisées par toutes les tâches sont connues a priori (contrairement à PIP)
- Définitions
  - Priorité plafond (priority ceiling) d'une ressource $\Pi(R)$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

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

- Protocole
  - Règle d'ordonnancement: Exécution selon priorité nominale, sauf règle 3 (idem PIP)
  - Règle d'allocation de la ressource R demandée par une tâche T
    - Ressource occupée : blocage (idem PIP)
    - Ressource libre
      - T strictement plus prioritaire que $\underline{\Pi}(t)$, allocation de R à T
      - Sinon, R allouée à T si c'est T qui possède la ressource de plafond $\underline{\Pi}(t)$, sinon, T est bloquée (différent PIP)
  - Règle d'héritage de priorité : qd blocage d'une tâche T lors de l'accès à R, la tâche bloquante Tl 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, Tl reprend la priorité qu'il avait à quand il a acquis la ressource.

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'
  - Π(R) = prio(T1); Π(R') = prio(T2); Initialement Π(0) = Ω;



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 Ti = Durée de la plus longue section critique d'une tâche de priorité inférieure et gardée par un sémaphore s tel que Π(s) ≥ prio(Ti)
    - Identifier les sections critiques des tâches de priorité inférieure qui peuvent bloquer Ti (direct + push-through)
    - Filtrer toutes les sections critiques vérifiant Π(s) ≥ prio(Ti) et prendre le maximum
  - Durée de blocage maximal Bi 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

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

- 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

  - 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

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), T4(6)
    - après filtrage, toutes les SC répondent à la condition
    - D'ou 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'ou B3 = max (6,5,4) = 6;
  - T4 : B4 = 0;

Partage de ressources
Exemples de primitives de gestion des mutex

Primitive	OSEK / VDX	VxWorks
Initialisation	Aucune (allocation statique)	sid = semBCreate(options,initialState); sid = semMCreate(options);
Destruction	Aucune	

Acquisition	GetResource(resid);	semTake(sid,timeout);
		taskLock(); /* Disable context switch */
		intLock();

Libération	SetResource(resid);	semGive(sid);
		taskUnlock(); /* Unable context switch */
		intUnlock();

PCP (ceiling fixé à l'init)

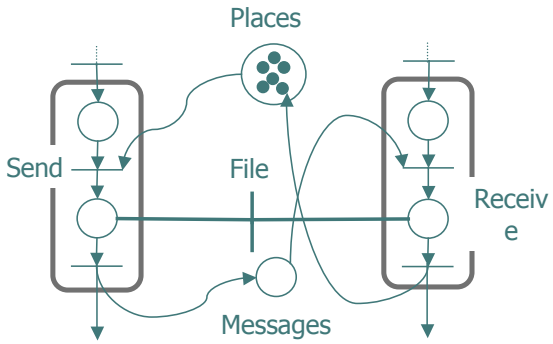
PIP et gestion de files (FIFO, prio. configurables)

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

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

- Interface typique :
  - Send(message,tâche);
  - Receive(message);
- Comportement



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

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;
```


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 la */
   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);
}
```

Communications
Exemples de gestion des boîtes à lettres

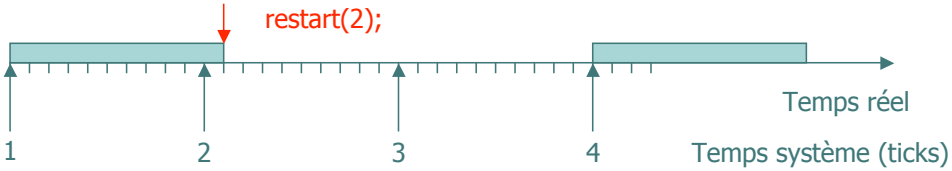
Primitive)	OSEK / VDX	VxWorks
Initialisation / création	InitMessage(mid,add);	mid = msgQCreate(maxMsgs, maxMsgLength, options);
Destruction	Aucune	msgQDelete (mid);
Envoi	SendMessage(mid, add); SendDynamicMessage(mid,add,length); SendZeroMessage(mid,add);	msgQSend (mid, buffer, nBytes, timeout, priority);
Réception	ReceiveMessage(mid, add); ReceiveDynamicMessage(mid,add,length);	msgQReceive(mid, buffer, maxNBytes, timeout);
Consultation	GetStatus(mid);	nmess = msgQNumMsgs(mid);

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)

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)

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 (cache quasi vide)

execs suivantes : 10 ticks (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

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);

Gestion du temps
Exemples de primitives

Primitive	OSEK / VDX	VxWorks

Exécution / événement non périodique	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	SetRelAlarm(aid,incr,cycle); SetAbsAlarm(aid,start,cycle);	
Annulation / destruction	CancelAlarm(aid);	timer_delete(tid);

Chiens de garde	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);

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)

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'ordonnançabilité
  - 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'ordonnançabilité

Gestion des interruptions
Primitives typiques

- Connection 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

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

Gestion des interruptions
Exemples de primitives

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

Gestion de la mémoire

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

Gestion de la mémoire
Statique vs dynamique

- Gestion statique de mémoire
  - Les nombres 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'execution (sûreté)
    - temps d'accès aux objets constant et faible (tableaux)
  - Inconvénients
    - Peu flexible

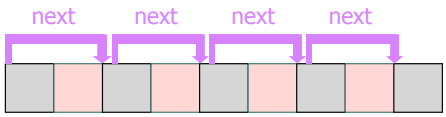
Gestion de la mémoire
Statique vs dynamique

- 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

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)
  - Chaînage de blocs libres dans les trous eux-mêmes
  - First-fit : on retourne le premier bloc libre de la liste de taille suffisante

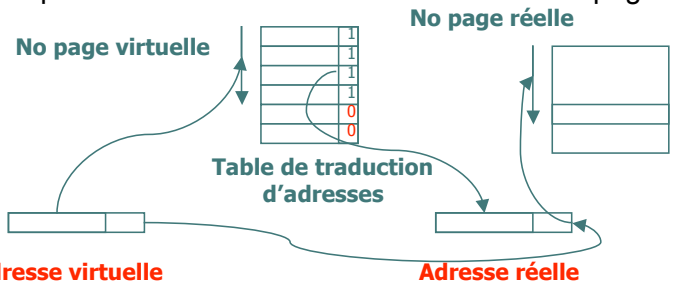
```
for (b = h->first;  b != tail ; b = b->next) {
  if ( b->size >= size ) {
    break; /* block found */
  }
 }
```



  - 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)

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)

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)

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;

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

Gestion de la mémoire
Exemples de primitives

Primitive	OSEK / VDX	VxWorks (non exhaustif)

Allocation	Aucune	ad = memalign (alignment, nbytes);
		ad = malloc(nbytes);
		ad = valloc(nbytes); /* *On page boundaries* */
		ad = calloc(nelem,elemsize);
		ad = realloc(adoldblock,newsize);

Libération	Aucune	free(ad); cfree(ad);

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
- Quelques exécutifs temps-réel industriels (liste non exhaustive …)
- Critères de sélection d'un exécutif temps-réel

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

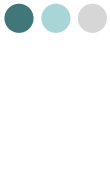
Extensions temps-réel de POSIX : 1003.1b
Ordonnancement

- 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 1iers ordos */
    - sched_getparam(pid,&param);
    - sched_setscheduler(pid,policy,&param);
    - sched_getscheduler(pid);
    - sched_yield();
    - sched_get_priority_max/min(policy);

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



Extensions temps-réel de POSIX : 1003.1b

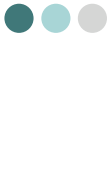
Communications

- 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 */

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);



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);

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 ?

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 debogueur
  - Simulateur (VxSim)
  - Observateurs très élaborés du comportement de la cible (WindView, StethoScope)
- Autre noyau du même vendeur : OSEKworks

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 supression des tâches en section critique
  - Sémaphores binaires : mutex sans ces 3 particularités

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

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, …)

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

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

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
  - Cibles supportées
  - Langages de programmation supportés (les plus courants : C,C++,Ada)
  - 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

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

- 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 ?
- 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

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

- 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
- Existence de drivers de périphériques
- Existence d'une chaîne de développement
- Support technique
- Services assurés par la société (portage sur nouvelles architectures)
- Nature du produit (code source ou binaire)
- Licence/prix (coût par système vendu ?)
- Réputation (nb de licences vendues, clients)
- Contraintes « métier » : certification

Vérification des contraintes de temps (analyse d'ordonnançabilité)

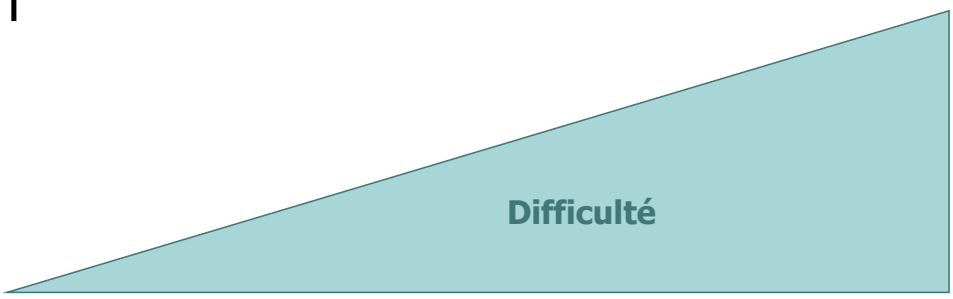
Vérification des contraintes temporelles
Introduction (1/2)

- 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

Vérification des contraintes temporelles
Introduction (2/2)

- Données d'entrée des méthodes de verification : 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)

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



Monoprocesseur
Tâches périodiques, synchrones, indépendantes

Multiprocesseur

Système distribué,
tâches périodiques et sporadiques, asynchrones, dépendantes

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

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

Monoprocesseur			Multiprocesseur		
Indep.	NoPr |Lmax	Poly	**NoPr, 2 proc**	NoPr,prec,Ci=1 | Lmax	Poly
	Pr | Lmax	Poly		NoPr,prec, ress	NP-hard
	NoPr,Ri | Lmax	NP-hard		NoPr, Ci <> 1 | Lmax	NP-hard
				NoPr, Ci=1, 1 ress	NP-hard
Préc.	NoPr,prec | Lmax	Poly	**NoPr, N Proc**	NoPr, prec, Ci=1	NP-hard
	NoPr, prec, Ri | Lmax	NP-hard			
	Pr, prec, Ri | Lmax	Poly			
Ress.	Per + sémas	NP-hard	**Pr, N proc**	Pr | mbmiss	NP-hard
	Pr,Ri,Ci=1 | Lmax	Poly			
	NoPr, prec, Ri, Ci=1 | Lmax	Poly			

Vérification de contraintes temporelles
Plan (1/2)

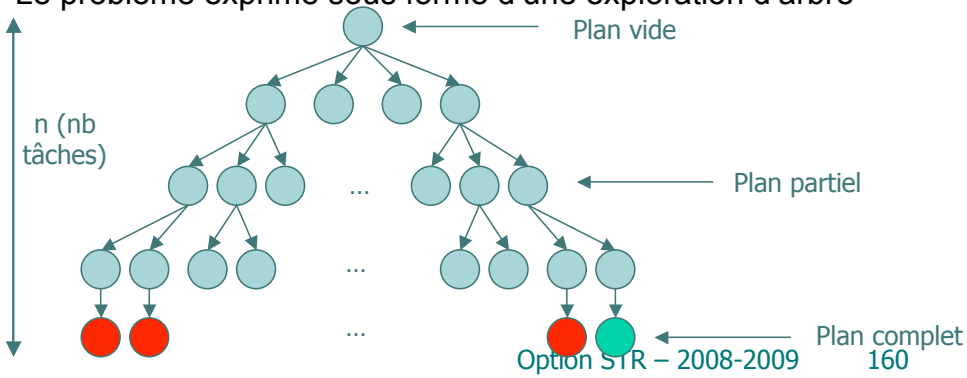
- Faisabilité pour tâches apériodiques (à dates d'arrivée connues) et ordonnancements non préemptifs
- Faisabilité pour tâches périodiques et ordos préemptifs à priorités
  - Rappel des notations et hypothèses
  - Approches par simulation
  - Condition nécessaire d'ordonnançabilité
  - Vérification pour 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 écheance ≤ période : Processor Demand Analysis (CNS)

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)

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



Faisabilité pour tâches apériodiques en non préemptif

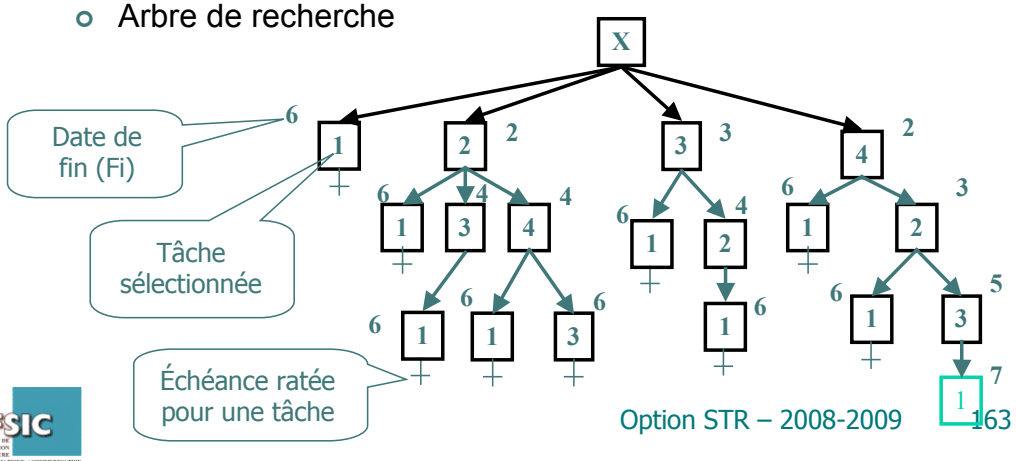
- Solutions
  - Exploration exhaustive de toutes les séquences d'ordonnancement possibles O(n n!) : trop complexe dans le cas général
  - Réduction de l'arbre de recherche
    - Elagage (pruning)
      - Exemple : algorithme de Bratley
    - Heuristiques pour guider le parcours de l'arbre
- Complexité de la recherche → orienté vers la vérification hors-ligne

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 noeud sur le chemin courant entraîne un dépassement d'échéance

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

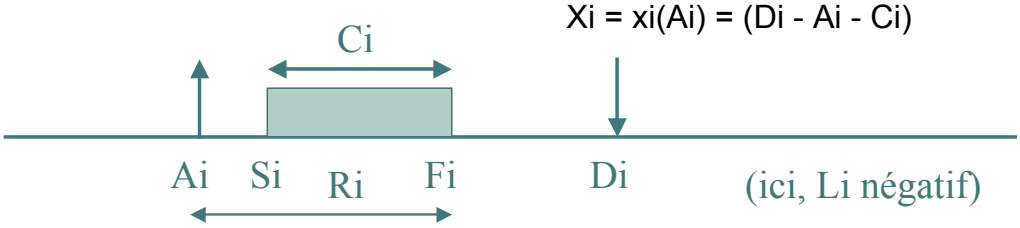


Faisabilité pour tâches apériodiques en non préemptif Algorithme de Bratley (1971) (3/3)

- 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 n!)
  - Utilisable pour de la vérification hors-ligne seulement

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

- Arrivée — Ai
- Temps de calcul maximum sans préemption — Ci
- Echéance (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)



Analyse d'ordonnançabilité 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

Analyse d'ordonnançabilité pour tâches périodiques
Hypothèses

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

Analyse d'ordonnançabilité 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

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

- Notion de charge processeur

$$U = \sum_{i=1}^{n} \frac{Ci}{Pi}$$

- Condition nécessaire d'ordonnançabilité : U ≤ 1
  - Cette condition n'est pas une condition suffisante (pas pour tous les ordonnancements)

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

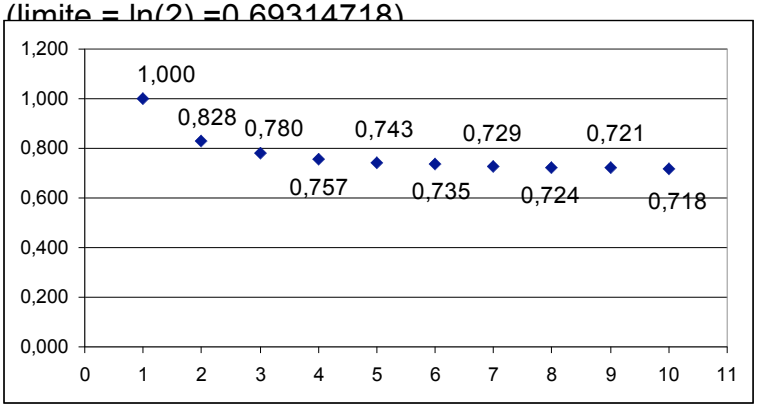
- Définition RM : la priorité d'une tâche est inversement proportionnelle à sa période (conflits résolus arbitrairement)
- Propriété RM : optimal parmi les algorithmes à priorités fixes pour des tâches périodiques indépendantes à échéance sur requête (Di=Pi).
- Condition de faisabilité

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

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

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

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



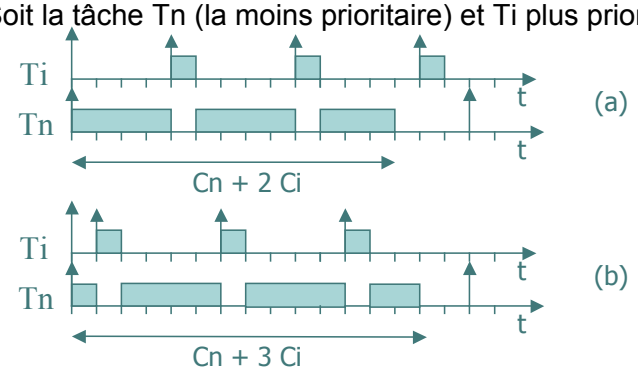
- Cas particulier : périodes des tâches sont des harmoniques (pour tout i,j tq Pi<Pj, Pj = k Pi avec k entier) : Ulub = 1

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

- Démarche
on montre que :
  - Instant critique pour une tâche Ti = quand la date d'arrivée de Ti 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'ordonnançabilité pour tâches périodiques
Rate Monotonic : preuve d'optimalité (2/6)

- Instant critique pour une tâche Ti = quand la date d'arrivée de Ti est simultanée avec les tâches de priorités supérieures
  - Soit la tâche Tn (la moins prioritaire) et Ti plus prioritaire que Tn



  - Si on avance la date Ai, la date de fin de Tn peut augmenter. La date de fin la plus grande est quand Ai = An
  - Transposable à toutes les tâches T1, …, Tn, d'ou le résultat

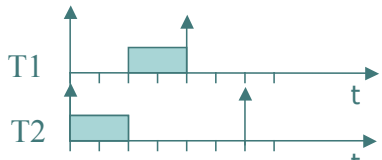
Analyse d'ordonnançabilité pour tâches périodiques
Rate Monotonic : preuve d'optimalité (3/6)

- Si jeu de tâches faisable avec des priorités arbitraires, l'est sous RM (démonstration pour n=2)
  - Deux tâches T1 et T2 avec P2 > P1
  - Démarche
    - Assignation des priorités selon des priorités arbitraires non RM (un seul choix pour 2 tâches !)
    - Etablissement de la condition de faisabilité dans ce cas (E1)
    - Assignation des priorités selon RM
    - On montre que si E1 vérifié, alors le système est faisable également avec l'assignation de priorités RM

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

- Assignation des priorités non-RM : prio(T_2) > prio(T_1)
  - On se place à l'instant critique
  - Le système est faisable si

$$C_1+C_2 \le P_1 \qquad (E1)$$

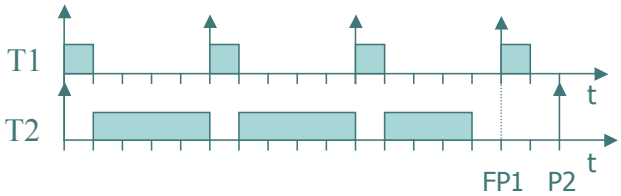


  - On montre que si E1 est vérifié, alors le système est aussi faisable en assignant les priorités selon RM
- Assignation des priorités selon RM : prio(T_1) > prio(T_2)
  - Soit F le nombre de périodes de T_1 intégralement contenues dans P_2

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

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

A C1 suffisamment court pour que toutes les instances de T1 arrivant dans l'intervalle [0,F P1] soient terminées avant la date P2



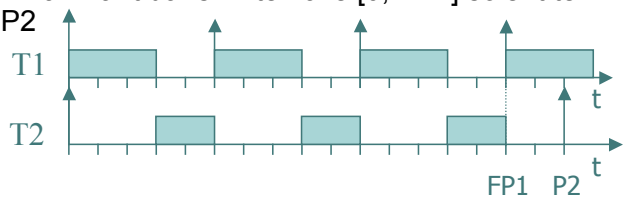
$$(A)\quad C_1 \le P_2 - F P_1$$

$$(F+1)C_1 + C_2 \le P_2 \qquad (E2)$$

- Système est ordonnançable si la condition (E2) est vérifiée (cf figure)
- On montre que si (E1) vérifié (non-RM), alors (E2) vérifié (RM)
  - D'après (E1) : $C_1 + C_2 \le P_1$, soit en multipliant par F : $FC_1 + FC_2 \le FP_1$
  - Comme $F \ge 1$, $FC_1 + C_2 \le FC_1 + FC_2 \le FP_1$
  - En ajoutant C_1 des deux cotés $(F+1)C_1 + C_2 \le FP_1 + C1$
  - Comme cas (A), $C_1 \le P_2 - FP_1$, d'ou $(F+1)\ C_1 + C_2 \le FP_1 + P_2 - FP_1$, soit $(F+1)\ C_1 + C_2 \le P_2$, ce qui vérifie (E2)

Analyse d'ordonnançabilité 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



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

- Système est ordonnançable 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'ou (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

Analyse d'ordonnançabilité pour tâches périodiques
Faisabilité pour Deadline Monotonic

- Hypothèse : hypothèses précédentes sauf H3 : Di ≤ Pi
- 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 ≤ période
- Condition de faisabilité

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

  - faible complexité O(n)
  - condition suffisante seulement (si Σ (Ci/Di) > U_{lub}, le jeu de tâches peut respecter ses échéances)
  - s'applique aussi quand les tâches ne sont pas synchrones

Analyse d'ordonnançabilité 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 Di (même si non inférieur à Pi : 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 Ri de chaque tâche à son instant critique (calcul itératif)
  - Vérification pour tout i que Ri ≤ Di

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

- Introduction au calcul de temps de réponse Ri

  Ri = Ci (temps d'exécution pire-cas de Ti)
  + Ii (interférences dues à des préemptions par tâches plus prioritaires)

  Nombre maximal de préemptions subies par une instance de Ti : $\sum_{j \in hp(i)} \left\lceil \frac{Ri}{Pj} \right\rceil$

  Interférence correspondante (Cj pour chaque préemption par Tj) : $Ii = \sum_{j \in hp(i)} \left\lceil \frac{Ri}{Pj} \right\rceil Cj$

  Temps de réponse Ri : $Ri = Ci + \sum_{j \in hp(i)} \left\lceil \frac{Ri}{Pj} \right\rceil Cj$
- Pas de solution simple (Ri apparaît des deux cotés)
  - Solution Ri = plus petite valeur telle que l'équation est vérifiée
  - Pas besoin de vérifier l'équation ∀ Ri ≤ Di (seulement quand le nombre de préemptions augmente)

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

- Algorithme de calcul des Ri (itératif)

$$w_i^0 = C_i$$
$$w_i^{k+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^k}{P_j} \right\rceil C_j$$

- Quand la série converge, on a calculé le temps de réponse Ri
- Le système est ordonnançable quand Ri ≤ Di

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

- Exemple

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

$$w_1^0 = C_1 = 2$$
$$w_1^1 = C_1 + \left\lceil \frac{C_1}{P_4} \right\rceil C_4 + \left\lceil \frac{C_1}{P_3} \right\rceil C_3 + \left\lceil \frac{C_1}{P_2} \right\rceil C_2 = 2 + \left\lceil \frac{2}{10} \right\rceil 3 + \left\lceil \frac{2}{5} \right\rceil 1 + \left\lceil \frac{2}{20} \right\rceil 2 = 2+3+1+2 = 8$$
$$w_1^2 = C_1 + \left\lceil \frac{8}{P_4} \right\rceil C_4 + \left\lceil \frac{8}{P_3} \right\rceil C_3 + \left\lceil \frac{8}{P_2} \right\rceil C_2 = 2 + \left\lceil \frac{8}{10} \right\rceil 3 + \left\lceil \frac{8}{5} \right\rceil 1 + \left\lceil \frac{8}{20} \right\rceil 2 = 2+3+2+2 = 9$$
$$w_1^3 = C_1 + \left\lceil \frac{9}{P_4} \right\rceil C_4 + \left\lceil \frac{9}{P_3} \right\rceil C_3 + \left\lceil \frac{9}{P_2} \right\rceil C_2 = 2 + \left\lceil \frac{9}{10} \right\rceil 3 + \left\lceil \frac{9}{5} \right\rceil 1 + \left\lceil \frac{9}{20} \right\rceil 2 = 2+3+2+2 = 9$$

Analyse d'ordonnançabilité pour tâches périodiques
Faisabilité pour EDF (Di=Pi)

- 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 ≤ période
- Condition de faisabilité (valide pour Di=Pi)

$$\sum_{i=1}^{n} \frac{Ci}{Pi} \leq 1$$

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

Analyse d'ordonnançabilité pour tâches périodiques
Faisabilité pour EDF (Di≤Pi)

- Conditions de faisabilité (Di ≤ Pi)
  - condition suffisante $\sum_{i=1}^{n} \frac{Ci}{Di} \leq 1$
  - condition nécessaire $\sum_{i=1}^{n} \frac{Ci}{Pi} \leq 1$

Analyse d'ordonnançabilité 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 Di = Pi
    - Expression de la demande en processeur
    - Notion de « busy period »
    - Condition de faisabilité
  - Extension au cas Di ≤ Pi

Analyse d'ordonnançabilité pour tâches périodiques
Processor demand - cas Di=Pi (2/6)

- Demande en processeur
  - Définition : demande en processeur sur [t,t+L] = quantité de calcul demandée par les Ti sur [t,t+L] qui doivent se terminer avant t+L
  - Demande en processeur sur l'intervalle [0,L] : $C_p(0,L)=\sum_{i=1}^{n}\left\lfloor\frac{L}{P_i}\right\rfloor C_i$
  - ⌊L/Pi⌋ est le nombre de fois ou la tâche Pi arrive dans l'intervalle [0,L] et doit se terminer dans l'intervalle (d'ou le ⌊⌋, on ne compte pas les arrivées se produisant dans l'éventuel morceau de période final)
  - Ci est son temps d'exécution à chaque lancement
- Le système est ordonnançable si pour tout L positif [Jeffay & Stone]

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

Analyse d'ordonnançabilité pour tâches périodiques
Processor demand - cas Di=Pi (3/6)

- Remarques
  - Ordo. cyclique : inutile de tester l'équation (e1) pour les L > PPCM(Pi)
  - Cp(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

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

  - ⌈L/Pi⌉ est le nombre de fois ou la tâche Pi arrive dans l'intervalle [0,L]
  - W(L) = charge de travail demandée dans l'intervalle [0,L]
  - Busy period Bp= min{ L | W(L) = L}
  - Busy period = période d'activité ininterrompue du processeur (peut être infinie si le système n'est pas ordonnançable)
  - Résultat : inutile de tester l'équation (e1) pour les L > Bp

Analyse d'ordonnançabilité pour tâches périodiques
Processor demand - cas Di=Pi (4/6)

- Calcul de la durée Bp de la busy period
  - Busy period Bp= min{ L | W(L) = L}

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

  - Algorithme (itératif)

```
L = Σ Ci; L' = W(L);
H = PPCM(P1, …, Pn)
while (L' != L && L' <=H) {
   L = L';
   L' = W(L);
}
if (L' ≤ H) Bp = L; else Bp = ∞;
```

- 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

Analyse d'ordonnançabilité pour tâches périodiques
Processor demand - cas Di=Pi (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,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 Di=Pi : on a déjà une condition nécessaire et suffisante, de complexité moindre (condition sur la charge)

Analyse d'ordonnançabilité pour tâches périodiques
Processor demand - extension au cas Di≤Pi (6/6)

- Demande en processeur dans le cas Di≤Pi

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

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

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

  - Calcul de la durée de la Busy Period identique au cas Di=Pi

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

	Di=Pi	Di ≤ Pi
Prio fixe	**Rate Monotonic** Faisabilité basée sur la charge (CS) $\sum_{i=1}^{n}\frac{Ci}{Pi}\leq n(2^{\frac{1}{n}}-1)$	**Deadline Monotonic** Faisabilité basée sur la charge (CS) $\sum_{i=1}^{n}\frac{Ci}{Di}\leq n(2^{\frac{1}{n}}-1)$ Response time analysis (CNS) - **toute priorité fixe** $R_i = C_i + \sum_{j\in hp(i)}\left\lceil\frac{R_i}{P_j}\right\rceil C_j \leq D_i$
Prio dyn.	**EDF** Faisabilité basée sur la charge (CNS) $\sum_{i=1}^{n}\frac{Ci}{Pi}\leq 1$	**EDF** 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
- Ordonnancements non préemptifs
  - Pas de problème (pas de préemption pendant les périodes d'utilisation des ressources)
- Ordonnancements préemptifs à priorités
  - Identification pour chaque tâche Ti d'une borne supérieure sur sa durée de blocage par des tâches de priorité inférieure (facteur de blocage) : Bi
    - Utilisation d'un protocole (PIP,PCP,SRP) nécessaire
  - Modification des conditions de faisabilité pour prise en compte des Bi

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

- Condition de faisabilité d'origine (CS) (Di=Pi) $\sum_{i=1}^{n}\frac{Ci}{Pi}\leq n(2^{\frac{1}{n}}-1)$
- 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'ou la condition $\sum_{i=1}^{n}\frac{C_i}{P_i}+\max(\frac{B_1}{P_1},\cdots,\frac{B_n}{P_n})\leq n(2^{\frac{1}{n}}-1)$

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

- Condition de faisabilité d'origine (CS) (Di ≤ Pi)

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

- 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

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

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

- Condition de faisabilité d'origine (CNS)

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

- Condition de faisabilité modifiée

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

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

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

- Condition de faisabilité d'origine (CNS) (Di=Pi)

$$\sum_{i=1}^{n} \frac{Ci}{Pi} \leq 1$$

- Condition de faisabilité modifiée (Di=Pi) - condition suffisante

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

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

- Condition de faisabilité d'origine (CS) (Di ≤ Pi)

$$\sum_{i=1}^{n} \frac{Ci}{Di} \leq 1$$

- Condition de faisabilité modifiée (Di=Pi) - condition suffisante

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

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

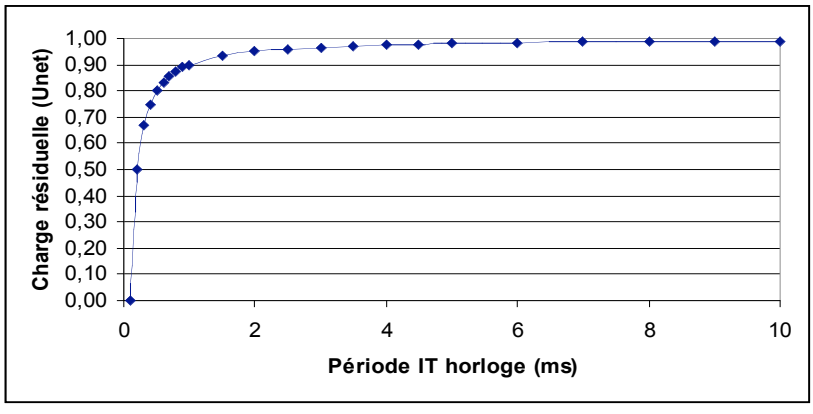
Prise en compte des coûts système
Introduction

- 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 (Ci)
  - 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 (Unet)



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 (Ni)
  - 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

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

  - Borne supérieure du nombre de préemptions subies par Ti dans le cas de tâches périodiques (bornes plus fines peuvent être trouvées selon l'ordonnancement)

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

Obtention des WCET (Ci)

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
- Points ouverts et recherches actuelles

WCET

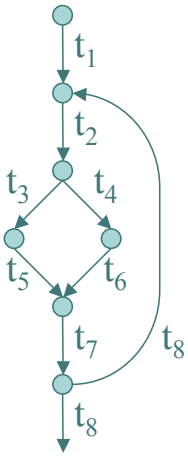
Définition

- 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

WCET

Eléments influençant le WCET

Programme

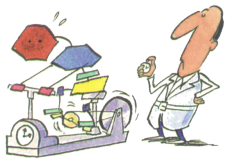


- Mises en séquence possibles des actions du programme (chemins d'exécution)
  - Dépendent des données d'entrée
- Durée de chaque occurrence d'une action dans chaque chemin
  - Dépendant de l'architecture cible

Méthodes d'estimation des WCET

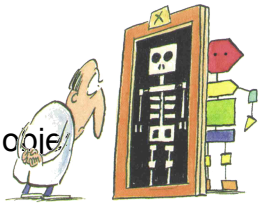
Méthodes dynamiques

- 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

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

Méthodes dynamiques vs méthodes statiques

Méthodes dynamiques (tests explicites)

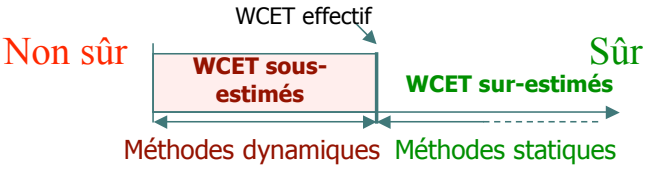
Temps mesuré ≤ WCET

WCET sous-estimé: le test de faisabilité peut accepter des ensembles de tâches non faisables

Méthodes statiques

Temps estimé ≥ WCET

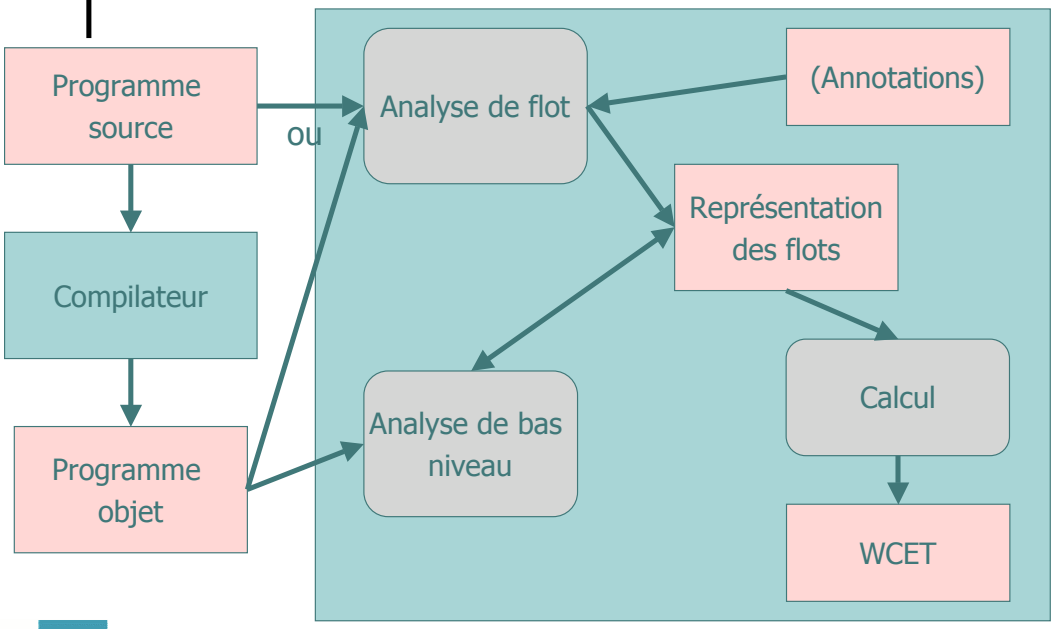
WCET surestimé: le test de faisabilité acceptera seulement des jeux de tâches faisables



Systèmes temps réel strict ont besoin de sûreté ⇨ Méthodes statiques

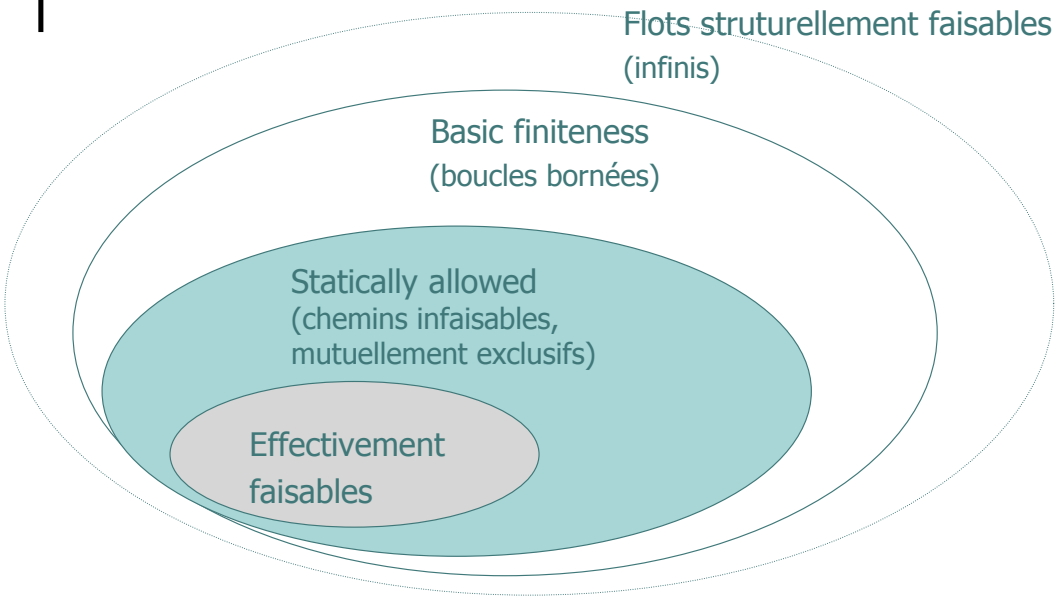
Méthodes statiques

Vue synthétique des différents composants



Méthodes statiques

Analyse de flot (1/3)



Méthodes statiques

Analyse de flot (2/3)

- Doivent être déterminés
  - Au minimum : nombres maximum d'itérations des boucles
  - Autres éléments pouvant améliorer la précision des WCET
    - Chemins infaisables ou mutuellement exclusifs
    - Bornes de boucles dans le cas de boucles non rectangulaires

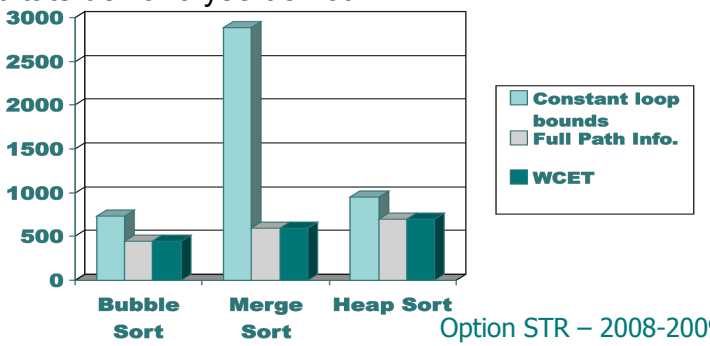
```
for i := 1 to N do                    ← Borne de boucle: N
   for j := 1 to i do                 ← Borne de boucle: N
   begin
      if c1 then A.long
            else B.short
      if c2 then C.short
            else D.long
   end
```

$\frac{(N+1)N}{2}$ executions

Méthodes statiques

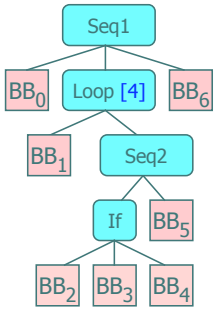
Analyse de flot (3/3)

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



Méthodes statiques : calcul

Analyse à base d'arbres (tree-based)

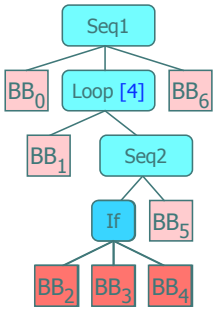


- Structures de données utilisées
  - Arbre syntaxique du programme
  - Blocs de base
- Principe de la méthode
  - Détermination des temps d'exécution des blocs de base (analyse de bas-niveau)
  - Calcul des temps d'exécution des chacune des structures syntaxiques du langage en utilisant les temps des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

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)



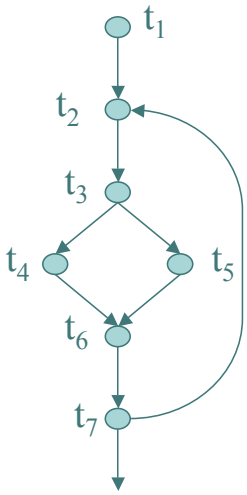
Système d'équations

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)

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)

Analyse IPET (Implicit Path Enumeration Technique)

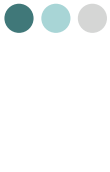


- Programmation linéaire en nombres entiers
  - But
    max: $f_1t_1+f_2t_2+\ldots+f_nt_n$
  - Contraintes structurelles
    $\forall v: \sum_{e_i \in In(v)} fa_i = \sum_{e_i \in Out(v)} fa_i = fv$
  - Contraintes sur les chemins
    exemples : $f_i \leq k$ (maxiter boucle)
    $f_i + f_j \leq 1$ (chemins mutuellement exclusifs)
    $\Sigma c_i f_i \leq k$ (ratio entre nombres d'exécutions)
- Programmation par contraintes
  - Expression de contraintes pour exprimer des relations plus complexes sur les chemins

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



Méthodes statiques : analyse de bas niveau

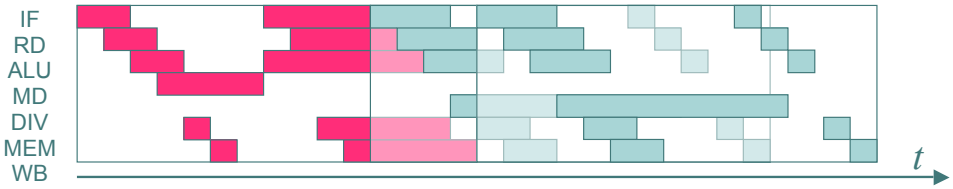
Introduction

- 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

Méthodes statiques : analyse de bas niveau (locale)

Prise en compte des pipelines

- Principe : parallélisme entre instructions sucessives (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



  - Obtenu 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

Méthodes statiques : analyse de bas niveau (globale)

Caches d'instructions

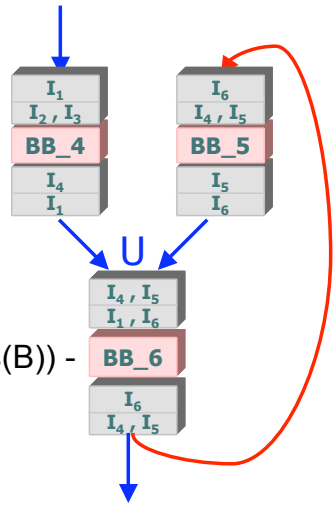
- Cache
  - Tire profit de la localité spatiale et temporelle des accès aux instructions
  - Spéculatif : comportement dépend du comportement passé des programmes
  - Bon comportement en moyenne, mais problème de déterminisme en contexte temps-réel strict
- Problème : estimation sûre du comportement 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)

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)

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, indépendant 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)

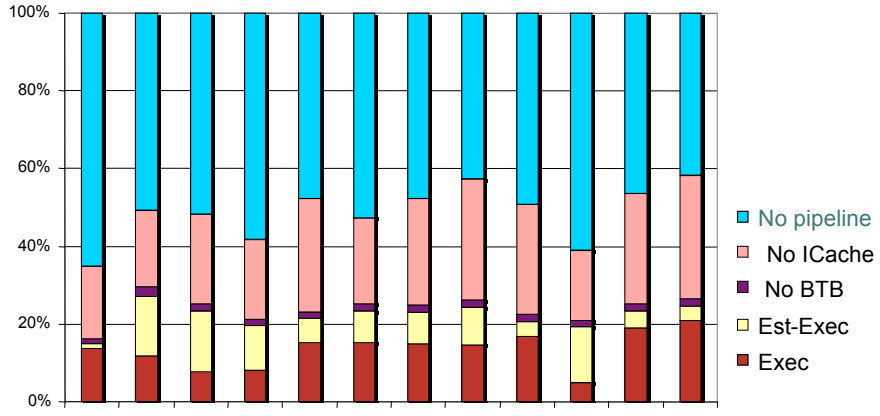
Autres éléments à prendre en compte

- 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

Méthodes statiques

Quelques résultats quantitatifs (Heptane)

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



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

Points ouverts et recherches actuelles

- Recherches actuelles : analyse de haut niveau
  - Détermination automatique des flots (Mälardalen)
  - 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, raffraichissement 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)

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 à des tâches qui ne sont pas indépendantes ou ont un Ci sous estimé

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

- 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

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 à 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 apériodiques, contraintes fermes, souples ou sans contraintes de temps
  - 2ème 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

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

Plan du chapitre

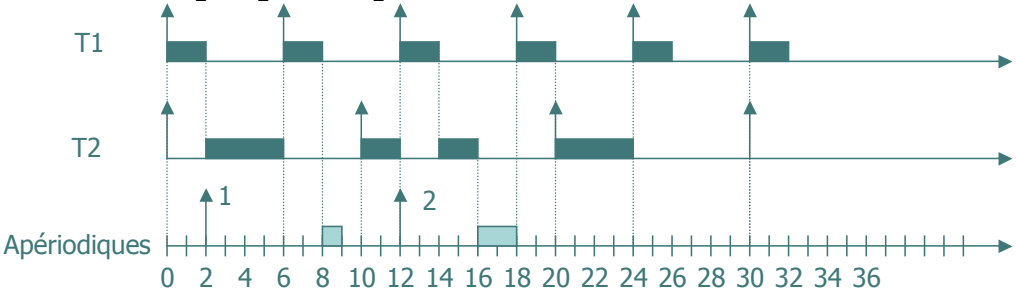
- Tâches apériodiques en tâche de fond (background scheduling)
- Gestion des tâches apériodiques par un serveur
  - Traitement par scrutation (polling server)
  - Serveur ajournable (deferrable server)
  - Echanges de priorités (priority exchange server)

Background scheduling (1/3)

- Principe
  - Exécution des tâches apériodiques quand aucune tâche périodique n'est prête à s'exécuter
- Exemple

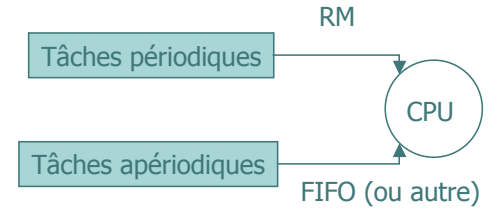
  $T_1 : C_1 = 2 \quad P_1 = 6$

  $T_2 : C_2 = 4 \quad P_2 = 10$



Background scheduling (2/3)

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



Mode de gestion des deux files indépendant, les files peuvent être gérées par des algorithmes d'ordonnancement différents

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

Serveur par scrutation (polling server) (1/8)

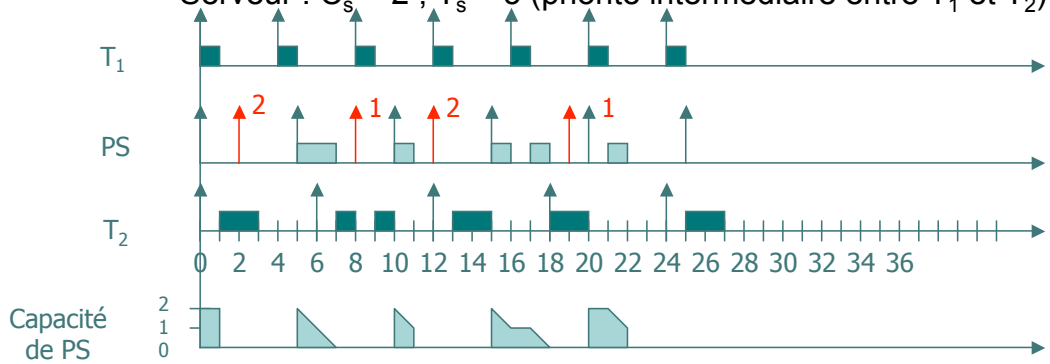
- Principe
  - Utilisation d'un serveur : tâche périodique dédiée à l'exécution des tâches apériodiques (principe commun à tous les ordonnanceurs à base de serveurs)
    - Tâche caractérisée par (C_s,P_s), Cs nommé capacité du serveur
    - Tâches apériodiques exécutées pendant au maximum C_s unités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

Polling server (2/8)

- Exemple de fonctionnement (ordonnancement RM)

$T_1 : C_1 = 1 \quad P_1 = 4 \qquad T_2 : C_2 = 2 \qquad P_2 = 6$

Serveur : $C_s = 2$; $T_s = 5$ (priorité intermédiaire entre T_1 et T_2)



Dates 1, 11, 22 : suspension du serveur (pas de tâches en attente de service)
Date 7 : suspension du serveur (capacité entièrement consommée)
Date 15 : préemption de PS par T1 : capacité conservée

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

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

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

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

Polling server (4/8)

- Condition pour la garantie d'une tâche apériodique (échéance ferme)
  - Condition (suffisante) pour une tâche isolée (i.e. pas d'autre tâche apériodique 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 : Ps (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 $P_s + \left\lceil \frac{C_a}{C_s} \right\rceil P_s \leq D_a$

Polling server (5/8)

- Remarque
  - La condition d'acceptation (garantie) d'une tâche apériodique 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)

Polling server (6/8)

- Condition de garantie d'une tâche apériodique : tâches non isolées, serveur de plus haute priorité
  - Tâches apériodiques 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 $D_i \leq D_k$

$$C_{ape} = \sum_{i=1}^{k} res_i(t)$$

$res_i(t)$ = temps résiduel à exécuter par la tâche apériodique i

  - 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.

Polling server (7/8)

- Date de fin d'une requête Jk
  - $F_k = t + C_{ape}(t,D_k)$ si $C_{ape}(t,D_k) \leq C_s(t)$
  - $F_k = (N_k + G_k)\, P_s + Res_k$ sinon

Avec N_k nb périodes entières pour exec, G_k n_o période de prise en cpte

$$N_k = \left\lfloor \frac{C_{ape}(t,D_k) - C_s(t)}{C_s} \right\rfloor$$

$$G_k = \left\lfloor \frac{t}{P_s} \right\rfloor$$

$Res_k = C_{ape}(t,D_k) - C_s(t) - N_k\, C_s$

- Condition d'acceptation d'une tâche apériodique : $\forall\, k \in [1,n_a]$, avec n_a le nombre de tâches apériodiques, 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 apériodiques en cours d'exécution

Polling server (8/8)

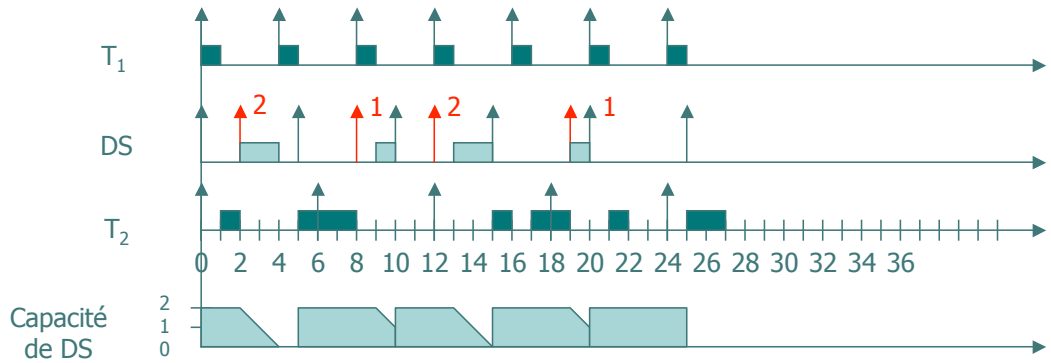
- Avantages
  - Reste simple, au niveau :
    - Des conditions de faisabilité pour les tâches périodiques
    - De l'acceptation en-ligne des tâches apériodiques
  - Meilleur service pour les tâches apériodiques que la gestion des tâches apériodiques en tâches de fond
- Inconvénients
  - Capacité du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

Serveur ajournable (Deferrable Server) (1/5)

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

Deferrable Server (2/5)

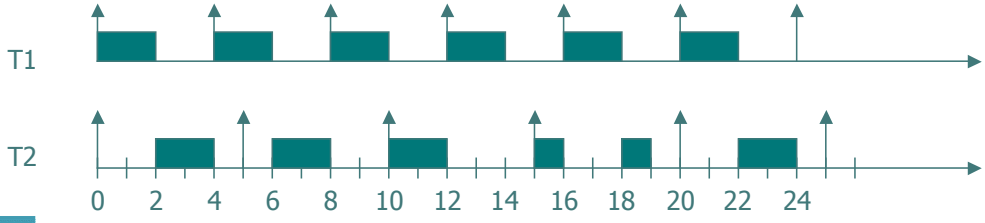
- Exemple de fonctionnement (même exemple que pour PS)



Dates 2,9,13, 19 : exécution d'une tâche apériodique (capacité conservée)

Deferrable Server (3/5)

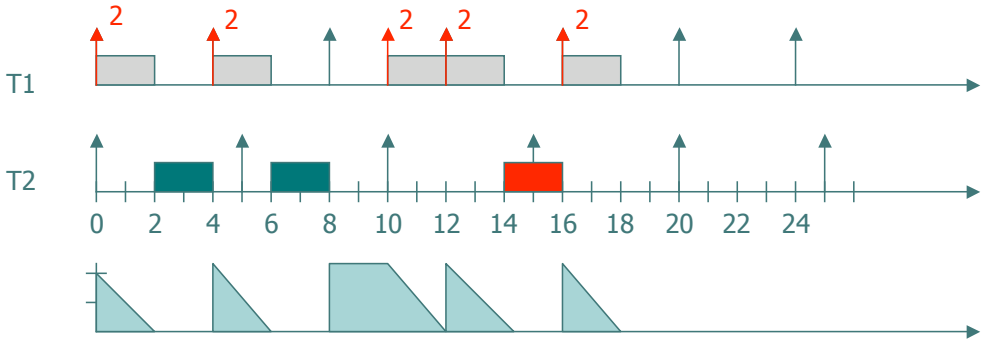
- Condition de faisabilité pour les tâches périodiques
  - Le serveur DS n'est plus équivalent à une tâche périodique vis à vis de la faisabilité du système (il ajourne son exécution, ce que ne font pas les tâches périodiques)
  - La condition suffisante pour l'ordonnancement RM n'est plus applicable
- Illustration
  - Cas a. Deux tâches périodiques T1 et T2 ($C_1 = C_2 = 2$, $P_1 = 4$, $P_2 = 5$)



(système ordonnançable sous RM)

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

Deferrable Server (5/5)

- Condition de faisabilité pour les tâches périodiques
  - En notant Us = (Cs/Ps), le système est faisable si la condition suivante est vérifiée (condition suffisante)

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

- Condition d'acceptation des tâches apériodiques
  - Existe
  - Même complexité algorithmique que PS
  - Accepte plus de tâches apériodiques que PS

Serveur à échange de priorités (Priority Exchange Server) (1/3)

- Principe
  - Serveur dédié à l'exécution des tâches apériodiques (idem PS et DS)
    - Par la suite, on considère que le serveur a la plus haute priorité
  - Quand le serveur a une tâche a périodique a exécuter, l'exécute à la priorité du serveur (idem PS et DS)
  - Quand le serveur n'a pas de tâche apériodique 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

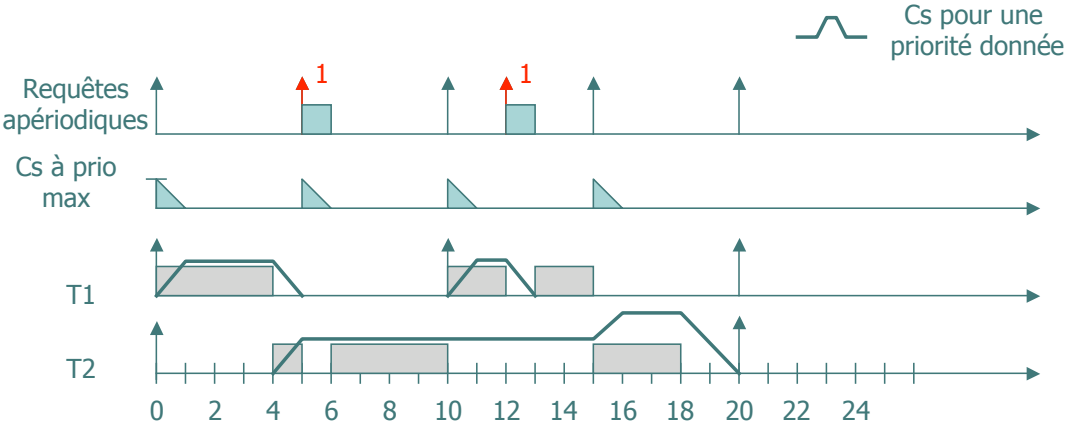
Ordonnancement hybride à base de serveurs

Priority exchange server (2/3)

- Exemple de fonctionnement

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

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



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)

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 ordonnancée par des algorithmes à priorités dynamiques (e.g. EDF)

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	🟠	🟠	🟠	🟠

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

Crédits : la majorité 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

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

Introduction

Evaluation stratégie (1/2) Utilisation composants [Lorch01]

Component	Hypothetical 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

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

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

Plan

- Introduction
- Les stratégies du système d'exploitation
  - concernant le support de stockage
  - concernant le processeur

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

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

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 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)

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

Support de stockage

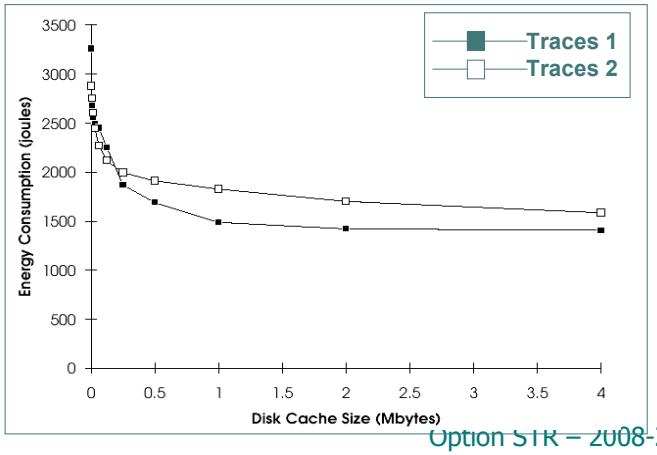
Transition – Mise en sommeil du disque (2/2)

- 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

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)



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

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)

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

Plan

- Introduction
- Les stratégies du système d'exploitation
  - concernant le support de stockage
  - concernant le processeur

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 µs à partir du mode *Deep sleep*, 10 cycles de bus à partir du mode *Auto halt*

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 : proportionel 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 ➔ "settling time" des portes plus important)
  - Intéressant quand la latence de changement de fréquence << latence de réveil du processeur

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

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]

Processeur

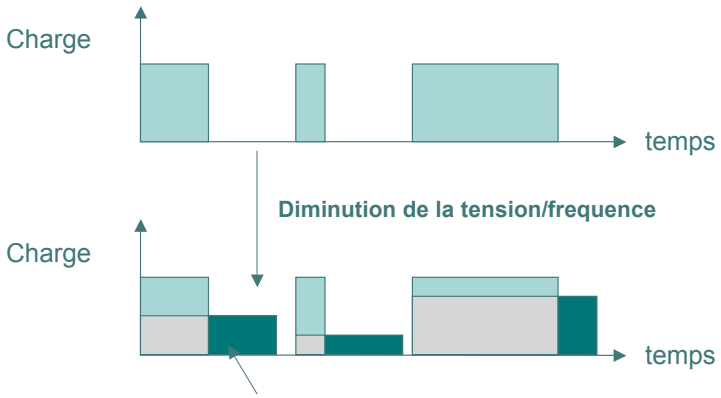
Load-change – Distribution et multiprocesseur (hétérogène)

- 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'ordonnancement

Processeur

Adaptation - changement de la fréquence du processeur



Cadencement plus faible,
Consommation beaucoup plus faible (quadratique / diminution tension),
Temps d'exécution plus long, impact sur le temps-réel,
Comme temps d'exécution plus long, le temps d'activation des autres composants est plus long

Processeur

Adaptation - changement de la fréquence du processeur

[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 croit, on augmente le cadencement du processeur,
  - Si l'utilisation décroit, 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

Processeur

Adaptation - changement de la fréquence du processeur

- [PBB98] Prédictions plus réalistes en agissant sur une fenêtre de temps plus longue. Différentes variantes :
  - *Past*, prédiction = activité période précédente
  - *Aged*, prédiction = moyenne des activité 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

Processeur

Adaptation - changement de la fréquence du processeur

[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
    - $U = \Sigma_i U_i$
  - Le test de faisabilité pour l'ordonnancement EDF est
    - $U = \Sigma_i U_i \leq 1$

Processeur

Adaptation - changement de la fréquence du processeur

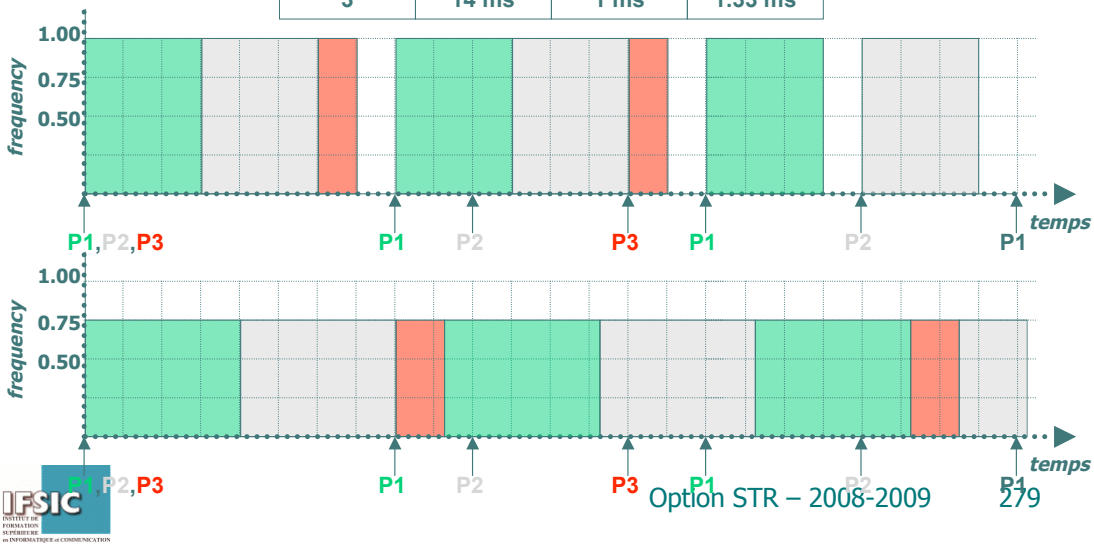
[PS01] ordonnancement DVS pour temps-réel strict

- Lors la fréquence du processeur est multipliée par un facteur α, 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 α
  - $U = \Sigma_i U_i \leq \alpha$
  - Possibilité de trouver la meilleure fréquence processeur telles que les échéances sont vérifiées
- Exemple pour un processeur disposant de 3 fréquences de cadencement α= (1, 0.75, 0.5)

Processeur

Adaptation - changement de la fréquence du processeur

Processus	Période	C α =1	C α =0.75
1	8 ms	3 ms	4 ms
2	10 ms	3 ms	4 ms
3	14 ms	1 ms	1.33 ms



Processeur

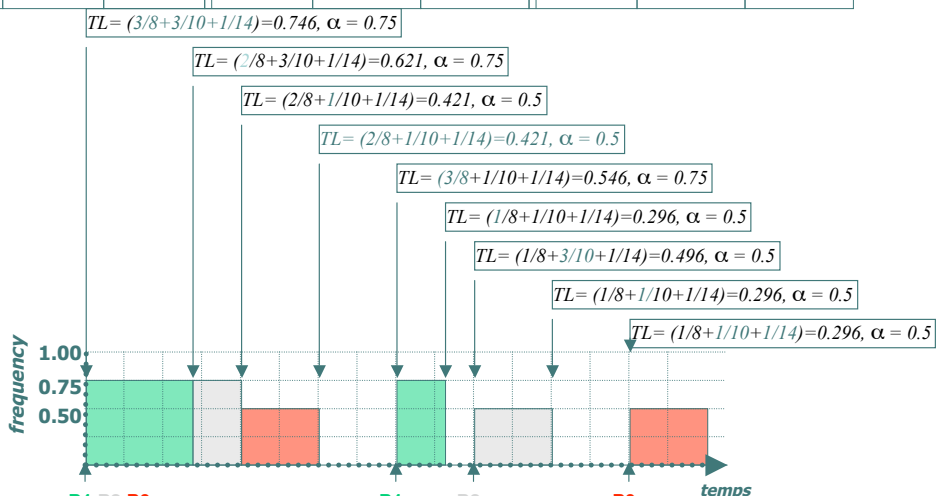
Adaptation - changement de la fréquence du processeur

- 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_i temps d'exécution 'réel'***
    - Ajuster l'utilisation à $U_i = cc_i/P_i$ pour **α=1**
  - Comme précédemment, choisir la fréquence pour que $U = \Sigma_i U_i \leq \alpha$,

Processeur

Adaptation - changement de la fréquence du processeur

Processus	Période	WCET α =1	Instance 1 α =1	Instance 1 α =0.75	Instance 1 α =0.5	Instance 2 α =1	Instance 2 α =0.75	Instance 2 α =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



Processeur

Adaptation - changement de la fréquence du processeur

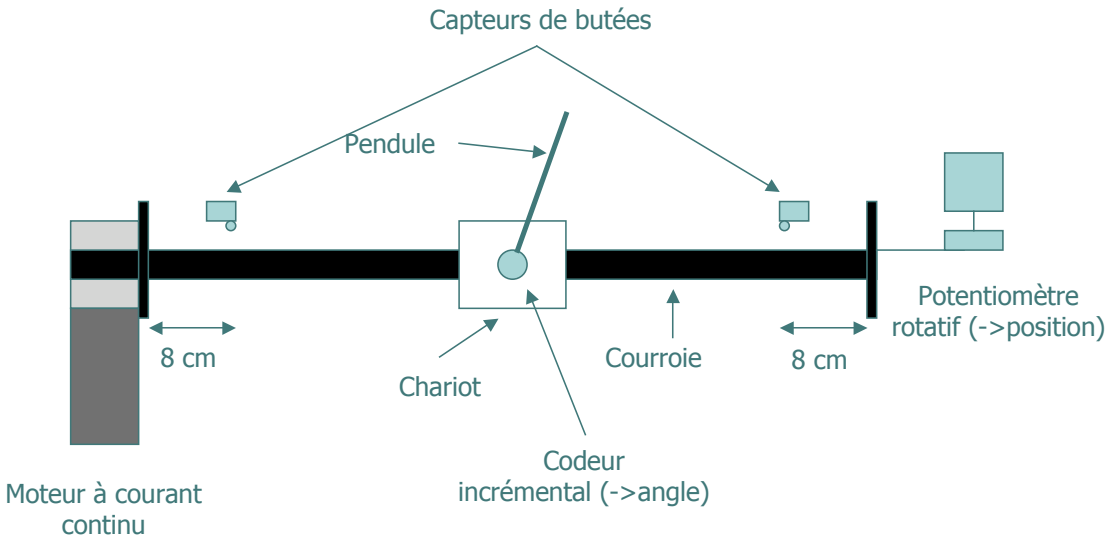
- Evaluations

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)

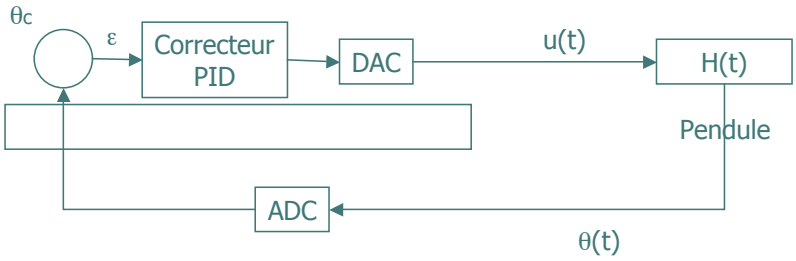
Etude de cas
Contrôle d'un pendule inversé

L'architecture



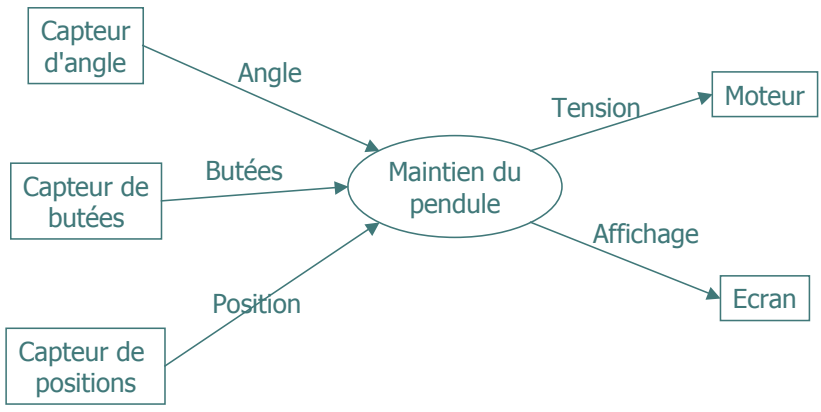
Contrôle du pendule

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



Système de contrôle

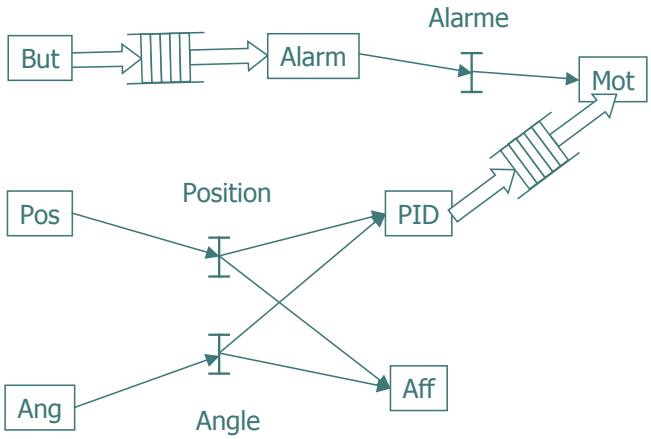
Diagramme de contexte de l'application



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 on 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)

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



NB : plusieurs schémas sont possibles

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 détaillé)
    - Délai de réaction = 10 ms
  - Régularité d'exécution de la tâche de contrôle du moteur

Détermination des paramètres temporels (2/3)

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

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

Ordonnançabilité

- 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