CEG4566/CSI4541 – Conception de systèmes temps réel

Chapitre 3 – Systèmes dynamiques, systèmes réactifs et systèmes à base de temps

3.1 Introduction à la dynamique des systèmes

• La dynamique d'un système traduit une abstraction du système et indique qu'il peut se trouver dans différents états, stables

• On étudie les conditions dans lesquelles, en fonction de différents événements, il passe d'un état à un autre.

• On s'intéresse à une dynamique discrète des systèmes (on parle de systèmes discrets). On parle d'aspects dynamiques *vs* aspects fonctionnels ou Statiques.

• Les systèmes dynamiques : Créer des processus, les finir, leur allouer des ressources et de la mémoire, etc. (Synchronisation, communication).

• La dynamique :

- Les réactions qui dépendent de l'état courant du système.
- Stimuli qui peuvent faire passer le système d'un état à un autre.

3.2 Les systèmes réactifs

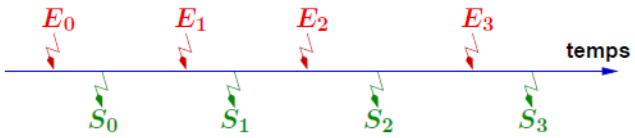
Les systèmes réactifs réagissent à différents stimulus de leur environnement et, en réponse, produisent des réactions, observables ou non.

Exemple *:* Réalisation d'un programme réactif simple

Étapes :

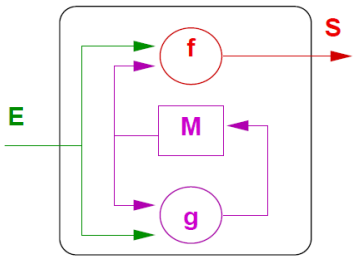
- Identifier :
  - Les entrées E, les sorties S
  - La mémoire interne nécessaire M, avec sa valeur initiale M0
- Définir :
  - La fonction de sortie Si = *f*(Mi,Ei)
  - La fonction de transition Mi+1 = *g*(Mi,Ei)
  - N.B. identique aux circuits dit séquentiels.
- Implémenter le tout par un programme

Implémentation simple : Géré par événement (*event-driven*)



Les E et les S alternent dans le temps
Le programme répond à Ei avant que n'arrive Ei+1
Le programme ne rate aucun changement significatif

```
Système(E, S)
   mémoire M
   M := M0
   boucle
      attendre(E)
      S = f(M, E)
      M = g(M, E)
      écrire(S)
   fin boucle
```



Ce système est un système temps-réel si : Le temps de calcul < temps de réaction de l'environnement.

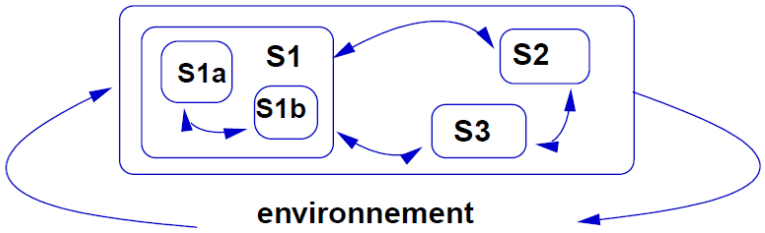
Implémentation simple : Échantillonnage

```
Système(E, S)
   mémoire M
   M := M0
   À chaque période faire
      lire(E)
      S = f(M, E)
      M = g(M, E)
      écrire(S)
   fin
```

Ce système est un système temps-réel si : Le temps de calcul < période, et période pour un environnement connu.

Cas Système réactif complexe : Découper le problème

- Gros système : beaucoup d'entrées/sorties
- Conception "mono-bloc" impossible
- Solution classique : conception parallèle et hiérarchique



Fonctionnement attendu : chaque sous-système se comporte localement comme un système "temps-réel".

3.4 Les aspects dynamiques des systèmes réactifs

- La prise en compte de la dynamique
- La prise en compte de la concurrence
- La prise en compte des communications et Synchronisations

3.5 Les langages et les modèles dynamiques

- Modèle états/transitions, discret
- Différents langages :
  - Proches du modèle (automates, réseaux de Petri, Statecharts, UML,...).
  - Plus abstraits (logiques, Algèbres de Processus)
- Deux axes complémentaires / duals
  - États d'abord (propriétés dans les états, logiques temporelles, etc.)
  - Transitions d'abord (étiquetage des transitions, abstraction sur les états, etc.)

3.6 Notion de Maitre-Esclave

- Ce principe est basé sur le principe qu'un thread (maitre) peut créer un autre thread (esclave) pour exécuter une tâche (par exemple une tâche d'E/S).
- Ce thread maitre continue après son exécution.
- Après, le maitre se synchronise avec l'esclave pour récupérer les résultats de la tâche.

Toute la question est : Comment éviter l'attente excessive de la part du maitre?

3.7 Systèmes à base de temps

3.7.1 Systèmes basés sur le temps *vs* basés sur les priorités

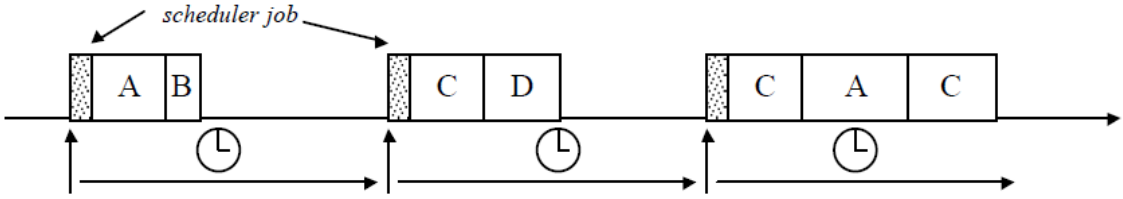
L'approche générale de l'ordonnancement temps-réel est souvent basée sur :

- Ordonnancement temporel (*Clock-driven* ou *time-driven*)
    - Les décisions d'ordonnancement sont faites à des intervalles de temps spécifiques qui sont choisi à priori.
- Ordonnancement par priorité
    - Les décisions d'ordonnancement sont faites quand un événement particulier arrive et que la tâche associée à cet événement est prioritaire et que le processeur est disponible.

3.7.2 Ordonnancement temporel (*Clock-driven*)

Temps de décision d'ordonnancement: C'est le point dans le temps quand l'ordonnanceur décide quelle tache va être exécutée prochainement.

- Défini à priori.
- L'ordonnanceur se réveille et génère une portion de l'ordonnancement.



Cas particulier: Quand les paramètres des tâches sont connus d'avance (à priori), l'Ordonnancement peut être calculé d'avance et mis dans une table (*table-driven*).

3.7.3 Ordonnancement par priorité

Règle générale: Ne jamais laisser le processeur à ne rien faire quand il y a un travail à faire (*work conserving*).

3.7.4 Ordonnancement dans les systèmes basés sur le temps

On peut appliquer les principes suivants;

- On calcule d'avance à l'aide d'algorithme lourd un ordonnancement statique.
- Tous les instants libres du processeur peuvent être utilisés pour exécuter des tâches apériodiques.

3.7.5 Implémentation

Table d'ordonnancement :

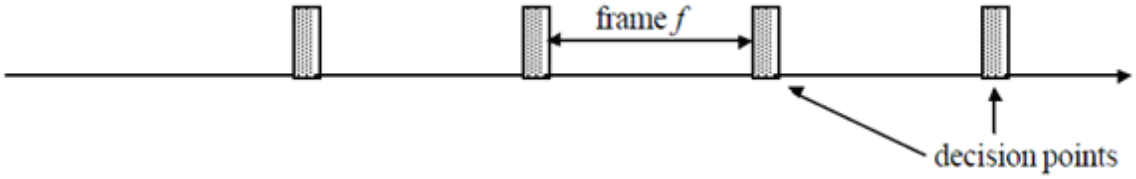
La table a des entrés de type (*tk, J(tk)*) , avec

- *tk* : temps de décision
- *J(tk)*: la tâche qui commence au temps *tk*
- *Entrée*: Ordonnancement (*tk, J(tk)*), $k = 0,1,\ldots,N-1$

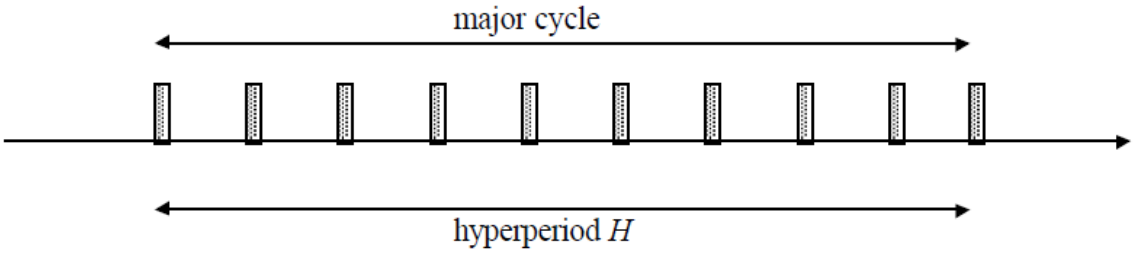
```
Task Scheduler:
    i := 0; k := 0;
    <set timer to expire at time t0>
    BEGIN LOOP
          <wait for timer interrupt>
          i := i+1;
          k:= i mod N;
          <set timer to expire at time
          (i DIV N)*H + t_k >
          IF J(t_k-1)is empty
          THEN wakeup(aperiodic)
          ELSE wakeup(J(t_k-1))
    END LOOP
END Scheduler;
```

Ordonnancement cyclique :

La décision d'ordonnancement est prise périodiquement. : Choisir la tâche à exécuter.



3.7.6 Notion d'hyper-période et de cycle majeur



Quelques contraintes:

- Les intervalles (frames) doivent être assez long pour permettre à chaque tâche de commencer et de se finir (mais pas trop long!), $f \geq \max(e_i)$
- L'hyper-période doit avoir un nombre entier d'intervalles.
- Pour des besoins de surveillance (moniteur), il faut prévoir suffisamment de temps dans un intervalle pour que la tâche et les autres utilitaires s'exécutent complètement (mais pas trop long!).

Que se passe-t-il pour les tâches apériodiques?

- Ordonnancement en background.
- Leur exécution pourrait être retardée

Note : Les tâches apériodiques sont généralement la résultante d'un événement externe. Donc, le système doit les exécuter le plus tôt possible (temps de réponse minimum). Rendre le temps de réponse le plus petit possible pour des tâches apériodiques est une difficulté de conception.

Solution : Exécuter les tâches apériodiques avant les tâche périodique chaque fois que c'est possible (*Slack Stealing*)

Que se passe-t-il pour les tâches sporadiques?

Rappel : Les tâches sporadiques sont définies comme des tâches où le début d'exécution et la durée d'exécution ne sont pas connus à priori.

On doit évaluer le temps du cas pire d'exécution (*WCET : Worst-Case Execution Time*)

Il faut faire un **test d'acceptante**

3.7.7 Pour ou contre l'ordonnancement temporel?

Pour :

- Conceptuellement simple
- Les contraintes temporelles peuvent être définies et vérifiées au niveau de chaque intervalle.
- On peut choisir la taille des intervalles qu'il faut.
- Facile à valider, le temps des intervalles et des hyper-périodes connu.

Contre :

- Maintenance difficile
- Ne permet pas d'intégrer les délais durs et mous.

3.7.8 Cas pire du temps d'exécution (*WCET : Worst-Case Execution Time*)

• On l'obtient soit par mesure, soit par analyse

• Le problème avec la mesure est qu'on n'est pas sûr est-ce vraiment le cas pire qui a été mesuré?

• L'analyse a besoin d'une bonne connaissance du processeur (cache, pipeline, *wait states*, etc.).

Comment calculer le WCET?

La plupart des techniques se reposent sur deux activités distinctes

- La première prend le processus et décompose son code en graph orienté formé de blocs basiques.
- Ces blocs représentent du code direct et simple à évaluer
- La deuxième activité de cette analyse prend le code machine correspondant à chaque bloc et utilise le modèle du processeur pour estimer le WCET.
- Quand le WCET de chaque bloc est connu, on peut alors évaluer le WCET du code complet.

Exemple:

```
for I in 1.. 10 loop
  if Cond then
    -- basic block of cost 100
  else
    -- basic block of cost 10
  end if;
end loop;
```

WCET=10*100 (+ ou -), on va dire 1000.

Mais si la condition *if* est 'vraie' uniquement 3 fois, alors WCET = 370 (+ ou -).

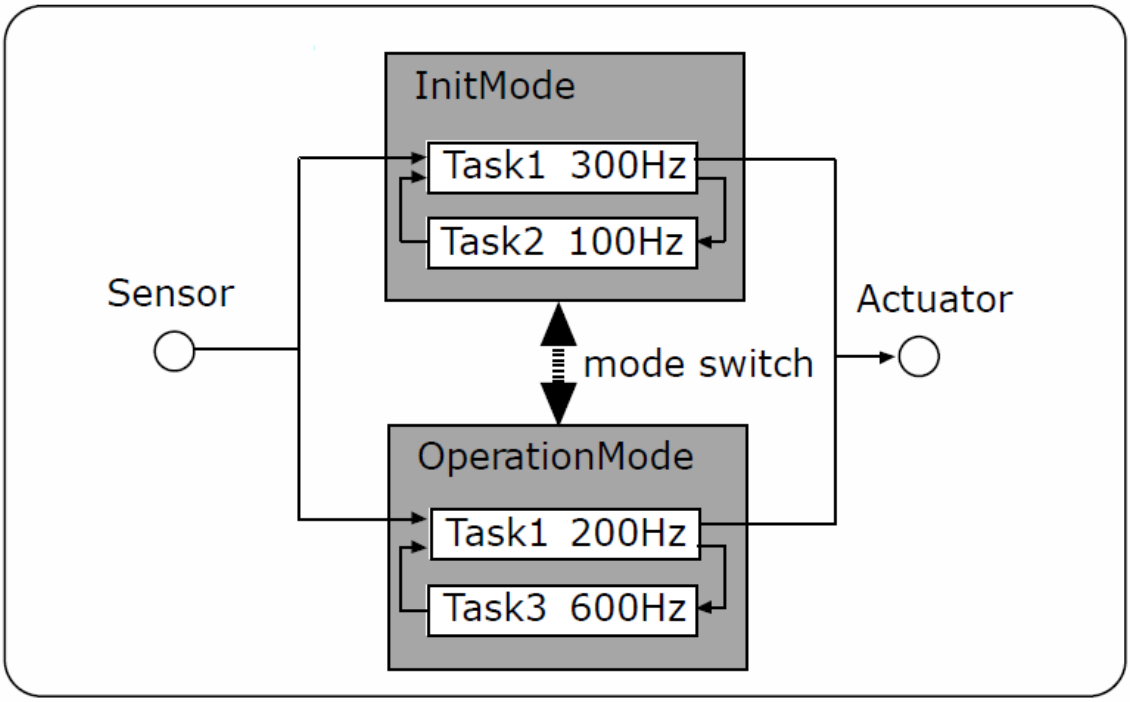
3.8 Les langages temporelles (TLD : *Time Definition Language*)

• Une notation textuelle de haut-niveau pour définir le comportement temporel d'une application temps-réel.

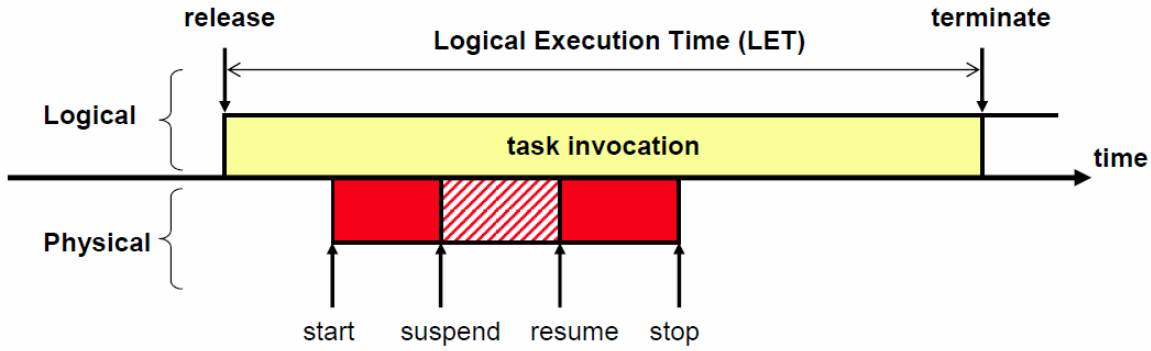
• Ce concept est basé sur GIOTTO (Université de Californie à Berkeley). Voir le lien suivant : http://www.eecs.berkeley.edu/Pubs/TechRpts/2000/CSD-00-1121.pdf

• TDL = Giotto + syntaxe + architecture + adaptations.

3.8.1 Le concept du Giotto/TDL



3.8.2 Le modèle de programmation du Giotto/TDL



On notera que ET ≤ WCET ≤ LET

Avec ET (*Execution Time*) et LET (*Logical Execution Time*)