Informatique Temps Réel - Résumé Janvier 2010.

Dubuc Xavier

7 décembre 2018

Table des matières

Introduction

Il s'agit d'un cours résumé servant uniquement **à l'étude**, car à l'examen, même si c'est «avec le cours», vous ne pourrez prendre ce pdf. Il est aussi à rappeller la répartition des points :

— 20% pour l'exercice de la taverne,
— 40% pour un exercice sur machine à l'examen (en rapport avec ce qui a été vu en **TP**),
— 40% pour la partie de l'examen concernant l'application de la théorie (cours ouvert).

1 Chapitre 1

1.1 Definitions

Un **système temps réel** est un système dans lequel l'exactitude des applications ne dépend pas seulement de l'exactitude des résultats mais aussi du temps auquel ce résultat est produit. Si les contraintes temporelles de l'application ne sont pas respectées, on parle de défaillance du système. Il est donc essentiel de garantir le respect des containtes temporelles du système.

Le système recueille des informations via des **capteurs** et il agit sur celui-ci via des **actuateurs**. Un exemple simple est un controleur de débit, on place un **capteur** calculant le débit à l'entrée d'un tuyau et plus loin on place un **actuateur** (une valve) servant à controler l'eau qui sort du tuyau.

Le système est un ensemble de **tâches** autonomes s'exécutant concurremment, le code d'une **tâche** correspondant à un bloc modulaire d'instructions. Chaque **tâche** possède son propre flot de contrôle à l'exécution et peut communiquer et partager ses ressources avec d'autres **tâches** du système. On définira donc des tâches associées à un ou des évènements, une ou des réactions, une entité externe à calculer, un traitement particulier, ...
L'**ordonnanceur** est le nom donné au module qui se charge de gérer l'enchaînement et la concurrence des tâches en optimisant l'occupation de l'unité centrale.

1.2 Fonctionnement

1. Fonctionnement cyclique

```
A chaque top horloge faire
   - Lecture de la memoire des entrees
   - Calcul des ordres a envoyer au procede
   - Emission des ordres
```

2. Fonctionnement évènementiel

```
while(1) {
   Attendre les evenements signales par interruption
   A chaque interruption faire {
      - Lecture information arrivee
      - Activation du traitement correspondant
      - Emission des ordres issus du traitement
   }
}
```

   Que faire si une interruption survient alors que le système est en train de traiter une interruption précédente ?
   → notion de priorité des interruptions
   → notion de tâche associée à une ou plusieurs interruptions
   → mécanisme de préemption (réquisition du processeur) et de reprise de tâche au retour du «vol» du **CPU**.
   → gestion de l'exécution concurrente des tâches (ordonnancement)

   Un système temps-réel fonctionnera de cette façon.

3. Fonctionnement mixte
   Mélange des 2 précédents.

1.3 Caractéristiques

Les caractéristiques nécessaires à un système T.R. sont les suivantes :
- **Fiabilité** (systèmes critiques)
- **Prédictibilité** (garantir le respect des contraintes temporelles)
- **Préemptibilité des tâches** (nécessaire pour la prédictibilité)
- **Gestion poussée des communications inter-tâches**
- **Prise en compte du non-déterminisme d'ordre d'exécution des tâches** ; on considère souvent le «worst-case» (cas au pire) pour déterminer un temps de réaction.

2 Chapitre 2 - Développement d'un système temps réel

2.1 Capturer les exigences

Analyse des besoins :
- Les **entrées** : *Acquisition de grandeurs par des senseurs.*
- Les **sorties** : *Commande de l'environnement par les actuateurs.*
- Les **traitements** : *Mise à jour des sorties en fonction des entrées pour contrôler/commander un processus physique.*
- La **visualisation** : *Présenter à l'utilisateur l'état du procédé et de sa gestion.*
- Les **communications** : *Partager des données avec les systèmes de contrôle commande voisins.*
- Les **sauvegardes** *des états pour des reprises d'exploitation ou des analyses du système a posteriori.*

1. Les tâches d'entrées
   Acquérir une donnée physique n'est pas trivial, il y a beaucoup de données de types différents et de capteurs de différentes natures et/ou ayant des fonctionnements différents.
2. Les tâches de sorties
   Il y a différents types d'actuateurs permettant au système d'agir sur l'environnement,
   - **Binaires** : soit on active soit on active pas, soit on ouvre soit on ouvre pas (enclenchement d'un ventilateur, ouverture d'une valve, ...),
   - **Discrètes** : sur quel type de bus,
   - **Continues** : des consignes précises quant à une modification (consigne de vitesse, température,...).
3. Les tâches de traitements
   Les tâches qui effectuent des analyses, des calculs sur les entrées afin d'en dégager les sorties.
4. Les tâches d'interface utilisateur
   Gérer un pupitre de commande permettant de visualiser l'état du procédé, les valeurs des consignes et les alarmes ainsi que d'agir sur les consignes et les commandes.
5. Les tâches de communication
   Tâches gérant la communication entre les tâches, elles sont contraintes par des contraintes logicielles et matérielles. (différents moyens de communiquer, différents moyens de formater les informations,...)
6. Les tâches de sauvegarde
   Elles sauvegardent les états du systèmes régulièrement afin de permettre d'analyser son fonctionnement, les erreurs, de permettre des reprises et mises en route ainsi que d'améliorer les performances.

Il faut maintenant parvenir à mettre en oeuvre l'ensemble de ces tâches en garantissant la **synchronisation**, les **transferts de données** et le **partage des ressources**. Pour ce faire, il y a 2 modèles :
- Synchrone
  - Les évènements émis par le procédé peuvent être *différés*.
  - Les évènements sont perçus comme *immédiats* par l'informatique.
  - Les évènements externes sont **synchronisés** avec les tâches.
  - Non préemptif, ordonnancement *à priori*, **séquenceur**.
- Asynchrone
  - Les évènements émis par le procédé sont traités immédiatement → interruptions.
  - Les évènements sont perçus comme *immédiats* par l'informatique.
  - Les évènements externes ne sont **pas synchronisés** avec les tâches.
  - Préemptif, ordonnancement *en ligne*, noyau **temps réel**.

2.2 Méthodologie stricte de développement

Il convient d'adopter une méthodologie pour le développement d'applications informatiques :
— L'architecture, la maintenabilité, la lisibilité, le suivi des spécifications, l'évolubilité sont augmentés.
— Nécessité de penser au préalable aux tests unitaires (par modules) et globaux.
Une méthodologie répandue, le **cycle en V** mais pas forcément adaptée au contrôle-commande :

```
Specification                                  Validation
      Conception Preliminaire              Integration
          Conception Detaillee    Tests Unitaires
                              Codage
```

Moins problématique, le **cycle en W** :

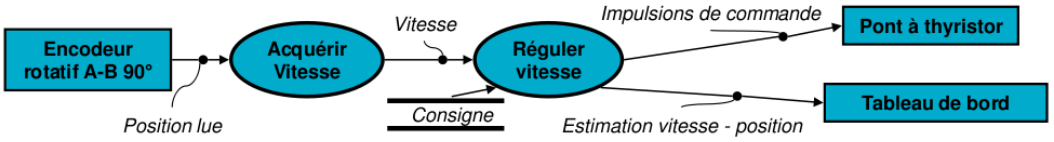
```
Specification                Simulation                                  Validation
     Conception      Integration  Conception Adaptee          Integration (avec noyau RT)
                  Tests                                  Tests
             Codage                          Codage (croise)
```

Nous allons utiliser 2 autres approches, plus complètes. Nous les développons ci-dessous.

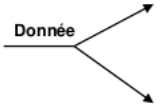
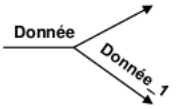
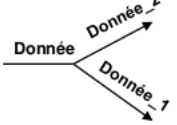
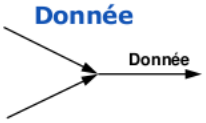
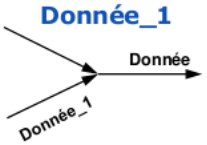
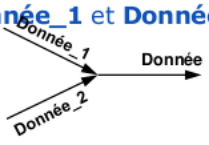
2.2.1 Structured Analysed Real Time (*SART*)

Éléments graphiques

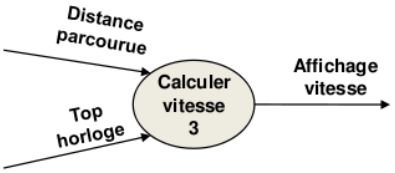
Fonction	Signification	Symbole	
Traitement - process	Unité de transformation de données d'entrée en données de sortie	-Cercle – bulle -Action: verbe+nom	Acquérir mesure
Flots de données	Vecteur nommé reliant deux process et véhiculant des données de même nature	-Flèches en trait plein -Données nommées	Mesures
Unité de stockage	Zone de rangement de données	-Deux traits // -Entité nommée	Mesures
Entité externe - terminateur	Source ou destination de données	-Rectangle -Entité nommée	Capteur



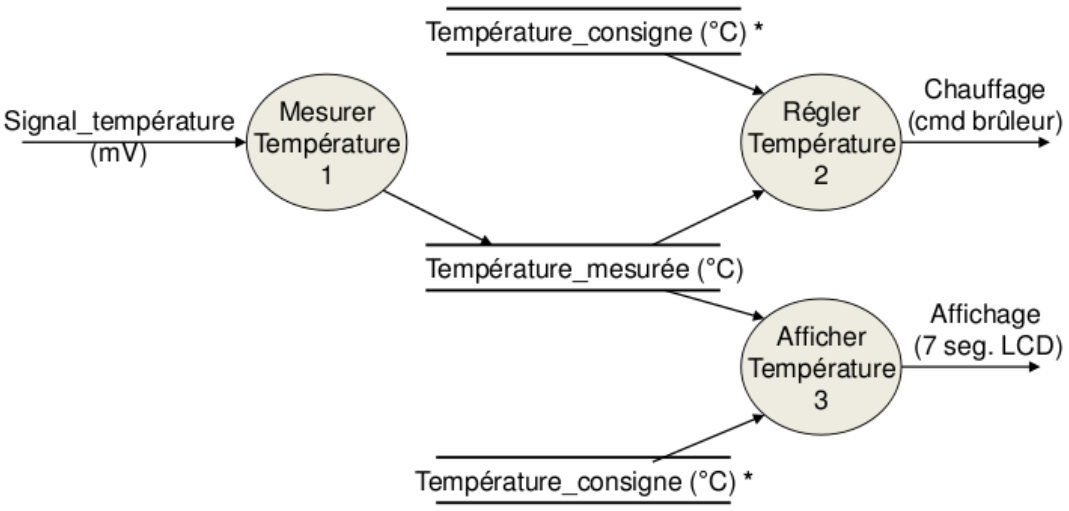
Les données

Donnée envoyée sur les deux flots	**Donnée_1** est extraite de **Donnée**	**Donnée_1** et **Donnée_2** sont extraites de **Donnée**
		
Création alternative de **Donnée**	**Donnée** est enrichie de **Donnée_1**	**Donnée** est construite par **Donnée_1** et **Donnée_2**
		

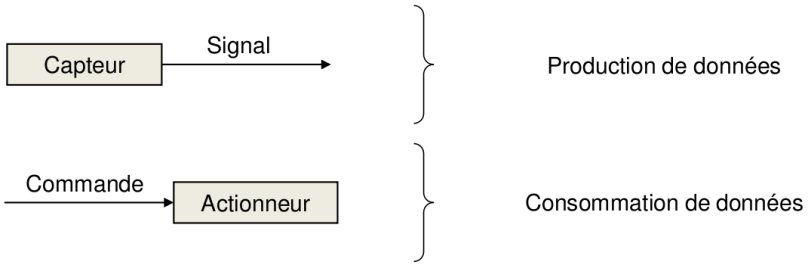
Les processus fonctionnels



Stockage



Terminaisons

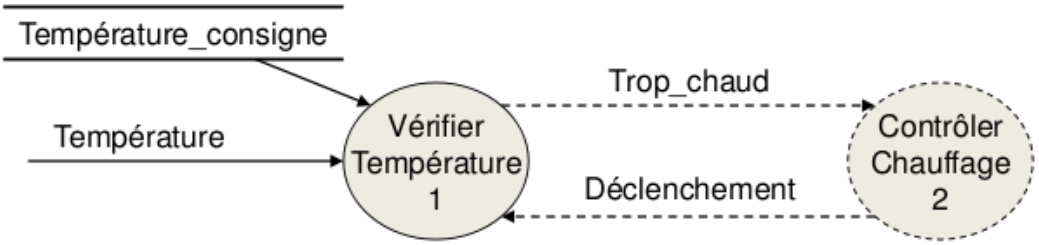


Les processus de contrôle

Le processus de contrôle représente la logique de pilotage des processus fonctionnels. Les processus fonctionnels fournissent tous les évènements aux processus de contrôle qui gèrent les évènements d'(de) (dés)activation des processus fonctionnels.

Les flots de contrôle

Les flots de contrôle transportent les évènements qui conditionnent, directement ou indirectement, l'exécution des processus fonctionnels.



Il y a 3 évènements prédéfinis : **E** *enable* (**activation**), **D** *Disable* (**désactivation**) et **T** *Trigger* (**enclenchement**).

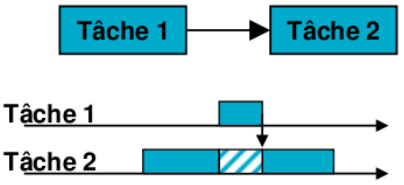
2.2.2 Design Approach for Real Time System (*DARTS*)

Cette approche permet de mettre l'accent sur la synchronisation et la communication entre les tâches.

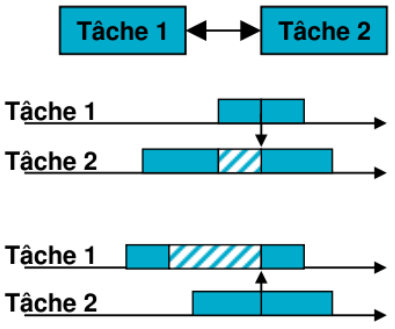
Relations entre tâches

Une tâche ne peut débuter que lorsqu'une autre tâche s'est exécutée en totalité ou en partie :

Asynchrone ou unilatérale : *une seule tâche est bloquée/en attente,*

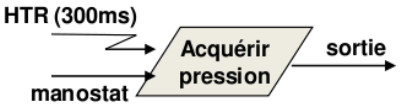


Synchrone ou bilatérale : *les 2 tâches doivent atteindre un point de rendez-vous,*

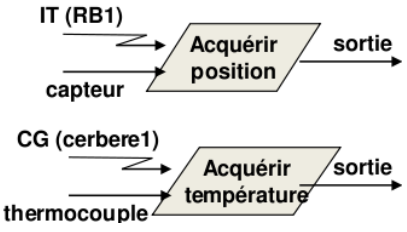


Activation des tâches

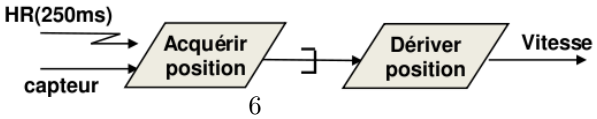
- Tâche matérielle :
  - périodique : horloge temps réel : **HTR***(durée)*



  - Non-périodique : interruption : **IT***(nom)* ou chien de garde : **CG***(nom)*

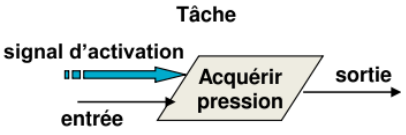


- Tâche logicielle : boîte aux lettres



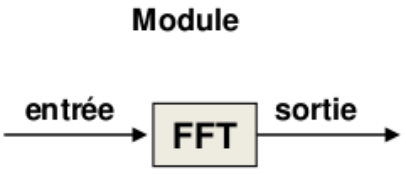
Tâches

Même définition que dans **SART** pour les processus fonctionnel mis à part qu'un signal d'activation est nécessaire en entrée.



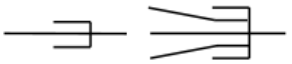
Module de traitement

Il s'agit de programmes spécifiques appelés par une ou plusieurs tâches, ils sont ré-entrants et permettent d'alléger le code des tâches tout en améliorant la structure.

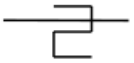


Synchronisation & Communication

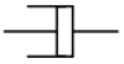
- **Synchronisation**
  - Asynchrone simple, multiple de type «*OU*»



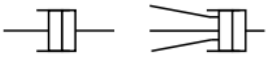
  - Synchrone



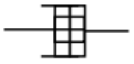
- **Communication** : boîte aux lettres (**BAL**)
  - **BAL** bloquante en écriture
    - 1 message



    - FIFO à n messages, simple ou multiples

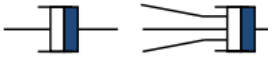


    - FIFO à n messages et priorités



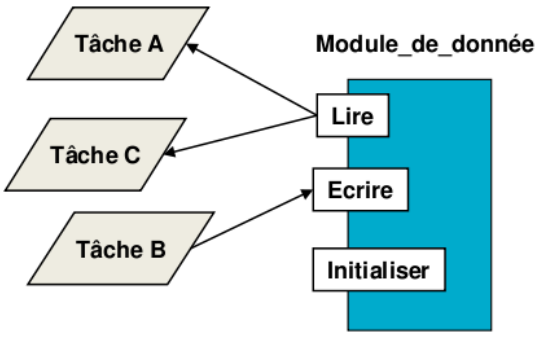
  - **BAL** non bloquante ou à écrasement
    - 1 message

    - FIFO à n messages, simple ou multiples



Stockage

Module de données avec protection des accès par exclusion mutuelle



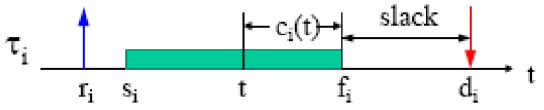
3 Chapitre 3 - Processus & Threads

Useless de résumer une notion vue & revue 42 fois en 3 ans ...

4 Chapitre 4 - L'ordonnancement

4.1 Table de correspondance des notations

Notation	Signification
τ_i	tâche de nom τ_i
$\tau_{i,j}$	job j de la tâche de nom τ_i
T_i	période de la tâcheΦ
r_i	temps de *release* ou d'arrivée dans le pool des tâches
s_i	temps de démarrage réel
Φ_i	temps de démarrage de la tâche (dans le cas de séparation en plusieurs jobs)
C_i	temps d'exécution au pire
d_i	deadline absolu
D_i	deadline relatif ($= d_i - r_i$)
f_i	temps maximal de fin d'une tâche (**contrainte** $f_i \leq d_i$)
$\max(0, f_i - d_i)$	avance ou retard de la tâche
$c_i(t)$	temps maximal restant à exécuter
$d_i - t - c_i(t)$	laxité, c'est-à-dire une période creuse d'exécution (**slack**)
p_i	Priorité de la tâche
R_i	Temps de réponse
U_i	Taux d'utilisation maximale du processeur par période par la tâche ($U_i = \frac{C_i}{T_i}$)
U_p	Taux d'utilisation totale du processeur $\left(U_p = \sum_{i=1}^{n} U_i\right)$



Dans le cours sont développées principalement 2 manières d'ordonnancer,

4.2 Tâches périodiques

4.2.1 EDF : Earliest Deadline First

$$\boxed{p_i = \frac{1}{d_i}} \textit{ (dynamique)}$$

Des tâches sont ordonnançables si : $\boxed{\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1}$

Demande du processeur dans $[0, L]$: $\boxed{g(0,L) \sum_{i=1}^{n} \left\lfloor \frac{L - D_i + T_i}{T_i} \right\rfloor C_i}$

L'ordonnancement est dès lors acceptable si $\forall L \in D,\ g(0,L) \leq L$.

où $D = \{d_k | d_k \leq \min(H, L^*)\}$, $H = lcm(T_1, ..., T_n) = ppcm(T_1, ..., T_n)$ *et* $L^* = \dfrac{\sum_{i=i}^{n} (T_i - D_i) U_i}{1 - U_p}$

4.2.2 RM (Rate Monotonic) (et DM (Deadline Monotonic))

$$\boxed{p_i = \frac{1}{D_i}} \textit{ (statique, souvent } D_i = T_i)$$

Des tâches sont ordonnançables si : $\boxed{\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1)}$

Temps de réponse : $\boxed{R_i = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_i}{T_k} \right\rceil C_k}$

Afin de calculer ce temps de réponse R_i, il existe une manière itérative d'y parvenir :

$R_{i,0} = C_i$

$R_{i,s} = C_i + \sum_{i=k=1}^{i-1} \left\lceil \frac{R_{i,s-1}}{T_k} \right\rceil C_k$

Calcul que l'on itère jusqu'à ce que $R_{i,s} = R_{i,s-1}$ ce qui signifie que l'on a trouvé le temps de réponse ou que $R_{i,s} > D_i$ qui signifie que le temps de réponse est plus long que le deadline relatif.

L'ordonnancement est dès lors acceptable si $R_i \leq D_i \ \forall$ tâche τ_i.