Systèmes temps-réels

Introduction

Matthieu Lemerre

March 5, 2008

Contenu du cours

- Cours 1: Qu'est ce que le temps réel?
- Cours 2: Parallélisme, communication et contrôle de la concurrence
  - Rappels d'architecture (notion de bus, interruption, atomicité, modèle von neumann)
  - Mécanismes de contrôle de la concurrence (sémaphores, spinlocks, instructions atomiques)
- Cours 3: Ordonnancement, analyse de faisabilité + fiabilité
  - Algorithmes d'ordonnancement
  - Guaranties de faisabilité
  - + Notions de fiabilités

Plan

Un système de traitement de l'information est appellé à remplir une mission:

- Mise en œuvre d'un certain nombre de *fonctions*,
- *suites* d'instructions codées (programme)
- des informations codées et organisées

Réalisation:

- Nécessite la mise en œuvre de ressources matérielles (HW, relativement figées)
- et de *logiciels* (MW et SW spécifiques)

Qualité des logiciels

Ils doivent être:

- **Efficaces** (réaliser les fonctions requises avec des performances *adaptées*)
- **Fiables** : Corrects, complets, sûrs
- **Testables** : compréhensibles, lisibles, structurés, auto-descriptifs
- **Portables** sur différentes machines
- **Maintenables** : corrections
- **Réutilisables** (évolutions)
- **Certifiables** (*prouver* le bon comportement)

Cela implique:

- Une conception conforme à la mission du système
- Une réalisation conforme à la conception

Faire une analyse des spécifications:

- Des méthodes et des règles
- des techniques élémentaires
- conception/programmation/mise en œuvre

Quelques chiffres

- Le logiciel représente 70% à 80% du coût (matériel $\approx$ 20%)
- 50% du budget sont pour les tests et la maintenance
- 60% des erreurs sont dues à la conception (80% en coût)
- 54% des erreurs sont detectées chez les clients

Aujourd'hui : coût estimé des bugs logiciels: 60 milliards $

L'approche modulaire

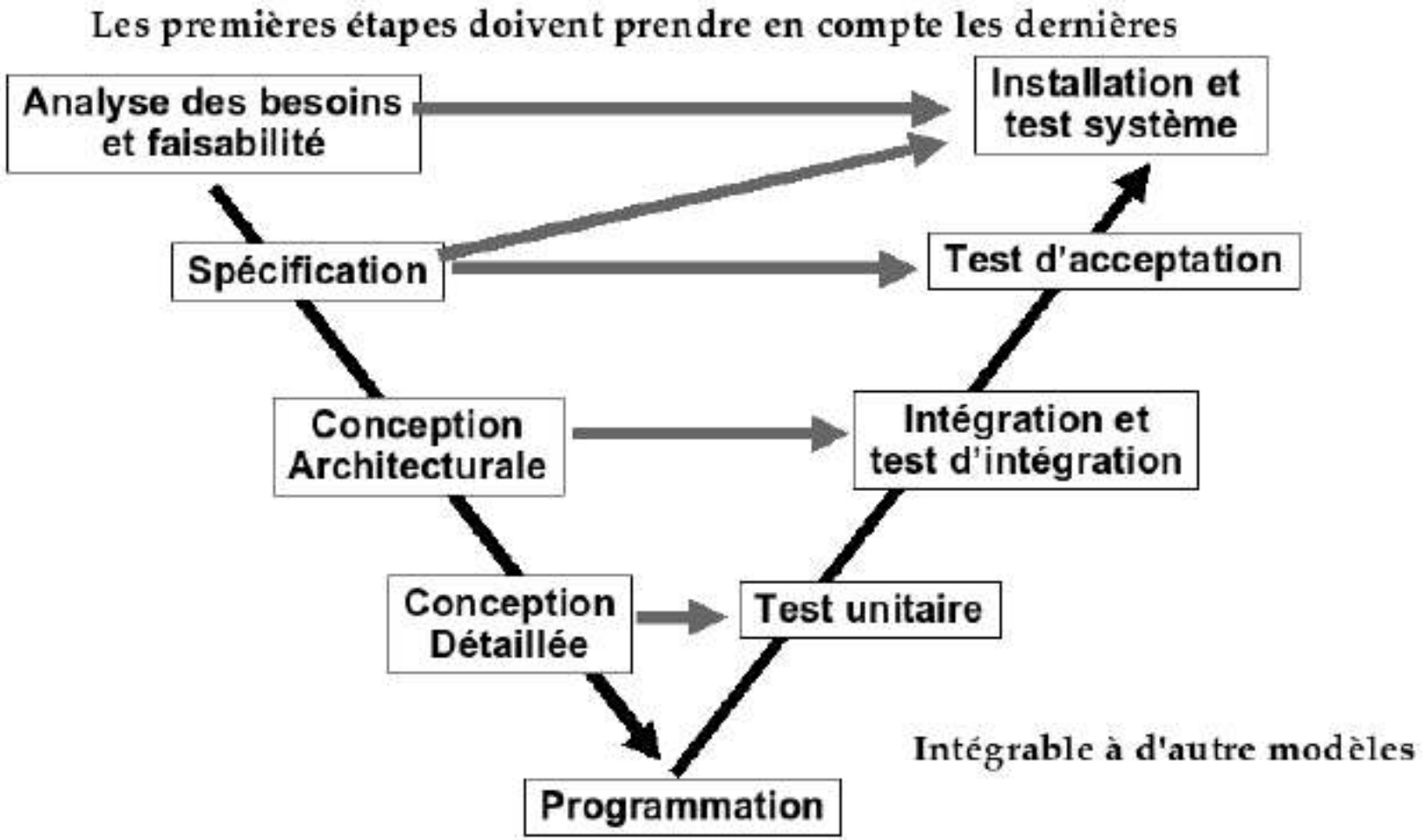
L'approche modulaire

- Permet la *division de la complexité* d'un problème
- Le *partage du travail* à réaliser

Les *méthodes* pour la cohérence des modules:

- Cohérence logique (répartatition par classe de problème, matériel ou logiciel)
- Cohérence temporelle (synchronisation, enchaînement des traitements)
- Cohérence procédurale (organisation des algorithmes)
- Cohérence par données communes (approche objet)
- Cohérence fonctionelle (1 fonctionalité donne 1 module)

Le modèle en V



Activités

Consensus général pour :

- Analyse des besoins
  - Experts du domaine d'application
  - Environnement, rôle, ressources, contraintes
- Spécification globale
  - Description de l'action du logiciel, pas de décision sur la réalisation
- Conception architecturale détaillée
  - Décomposition du logiciel, spécification des interfaces, description de la réalisation des composants
- Programmation
  - 15% de l'activité
- Gestion de configuration et intégration
  - Permettre l'évolution des composants, assemblage
- Vérification et validation
  - Adéquation aux besoins, satisfaction de la spécification
  - Analyses, tests

Plan

Definition (Système temps réel)

Un système temps réel est un système qui a la capacité de répondre à des évènements asynchrones issus du monde physique dans des délais pré-déterminés.

Système temps réel dur (Hard Real-Time System / Time-Critical Real-Time System):

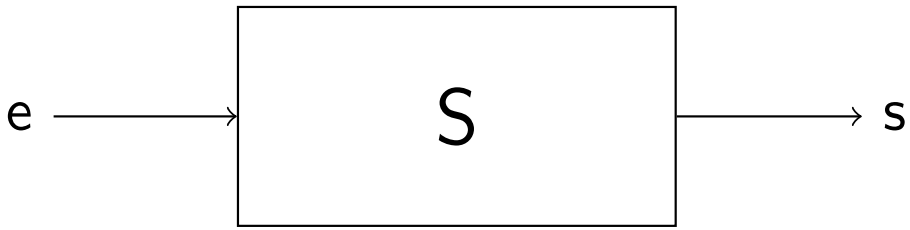
- Une contrainte non respectée est considérée comme une défaillance du système.
- Entraine un risque de dysfonctionnement, risque d'incident.

Life-Critical Real-Time System / Safety-Critical Real-Time System:

- Idem, mais incident $\longrightarrow$ accident

Attention: système temps réel $\neq$ rapide!

Système simple



- e : entrée du système S
- S : processus
- s : sortie du système S

$$s = F(e)$$

A priori, non temps-réel

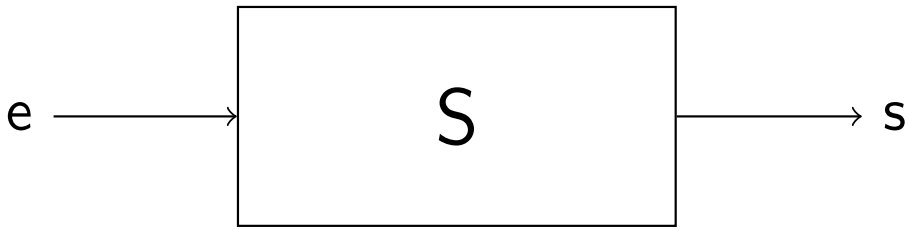
Analyse:

- Structure de données
- Algorithmes

$etat_1 \longrightarrow$ Puis le programme $\longrightarrow etat_2$

Aspects temps-réel?

Système simple



- e : entrée du système S
- S : processus
- s : sortie du système S

$$s = F(e)$$

A priori, non temps-réel

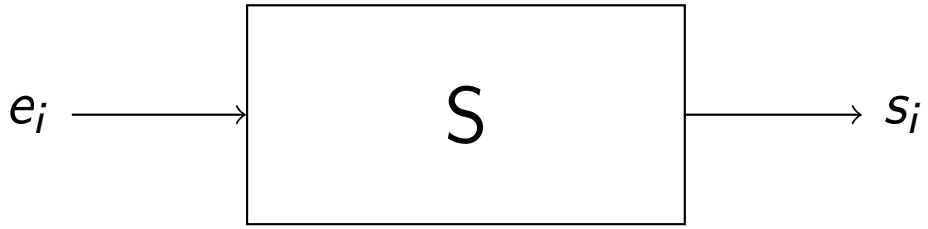
Analyse:

- Structure de données
- Algorithmes

$etat_1 \longrightarrow$ Puis le programme $\longrightarrow etat_2$

Aspects temps-réel?

Système classique



Multiple flots de données

- e_i : entrées du système S
- s_i : sorties du système S

$$s_i = F(e_i)$$

$$s_i = F(e_i, e_{i-1}, e_{i-2}), \ldots = F(E_{i-1}, e_i)$$

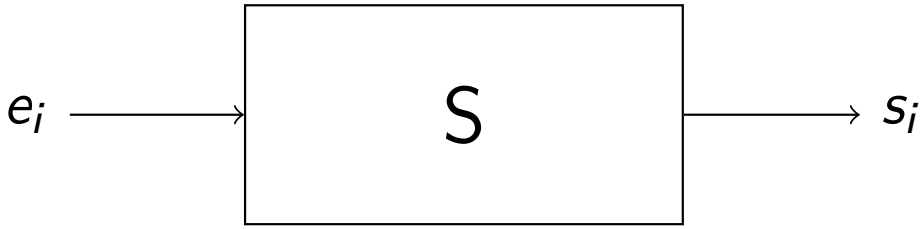
.

Caractéristiques:

- Non terminaison du système
- Cadencement événementiel
- Les traitements et les E/S se succèdent et s'entrelacent
- Le système peut *attendre* un évènement

Est-ce temps-réel?
(lois et temps de cycles)

Système classique



Multiple flots de données

- e_i : entrées du système S
- s_i : sorties du système S

$$s_i = F(e_i)$$

$$s_i = F(e_i, e_{i-1}, e_{i-2}), \ldots = F(E_{i-1}, e_i)$$

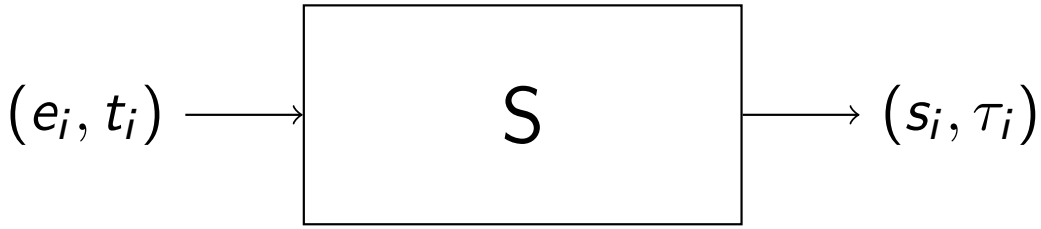
Caractéristiques:

- Non terminaison du système
- Cadencement événementiel
- Les traitements et les E/S se succèdent et s'entrelacent
- Le système peut *attendre* un évènement

Est-ce temps-réel?
(lois et temps de cycles)

.

Système temps-réel



Les flots de données suivent des *lois temporelles*

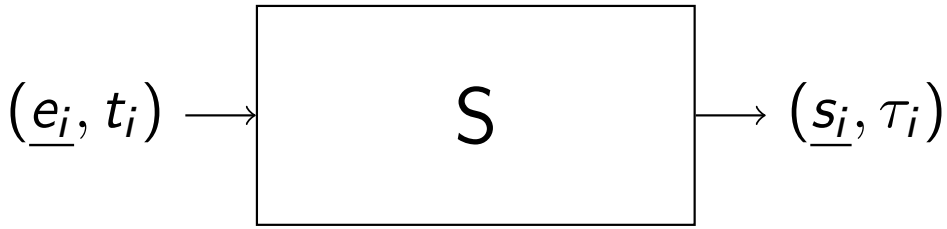
- (e_i, t_i) : événements reçus par le système S
- $s_i, \tau_i)$: événements émis par le système S

$$s_i = F(E_{i-1}, e_i, t_i)$$

$$\tau_i = G(E_{i-1}, e_i, t_i)$$

Si fonctionnelle...

Système temps-réel numériques



On passe d'un milieu *continu* (systèmes analogiques) à un milieu *discontinu* (systèmes numériques)
Les lois événementielles sont discrétisées:

- Effets souvent imperceptibles, mais réels...

$\Rightarrow$ Heisenbugs dans les systèmes *non déterministes* (i.e. non reproductibles par rapport aux conditions initiales et évenementielles)

.

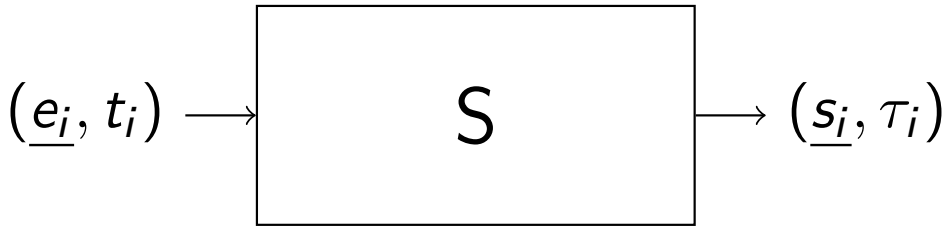
Conséquences de la numérisation:

- Entrées/Sorties approximatives
- Calculs exacts (!)

$\Rightarrow$ Meilleures précisions.
Applications directes:

- Traitement du signal
- Filtrage non linéaire
- Codage/décodage sans bruit
- Compression de données
- Cryptage

Système temps-réel numériques



On passe d'un milieu *continu* (systèmes analogiques) à un milieu *discontinu* (systèmes numériques)
Les lois événementielles sont discrétisées:

- Effets souvent imperceptibles, mais rééls. . .

$\Rightarrow$ Heisenbugs dans les systèmes *non déterministes* (i.e. non reproductibles par rapport aux conditions initiales et évenementielles)

.

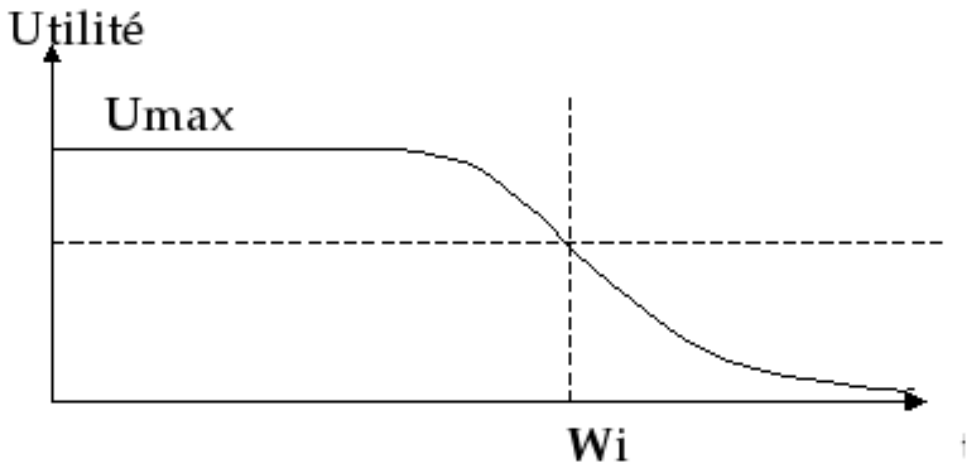
Conséquences de la numérisation:

- Entrées/Sorties approximatives
- Calculs exacts (!)

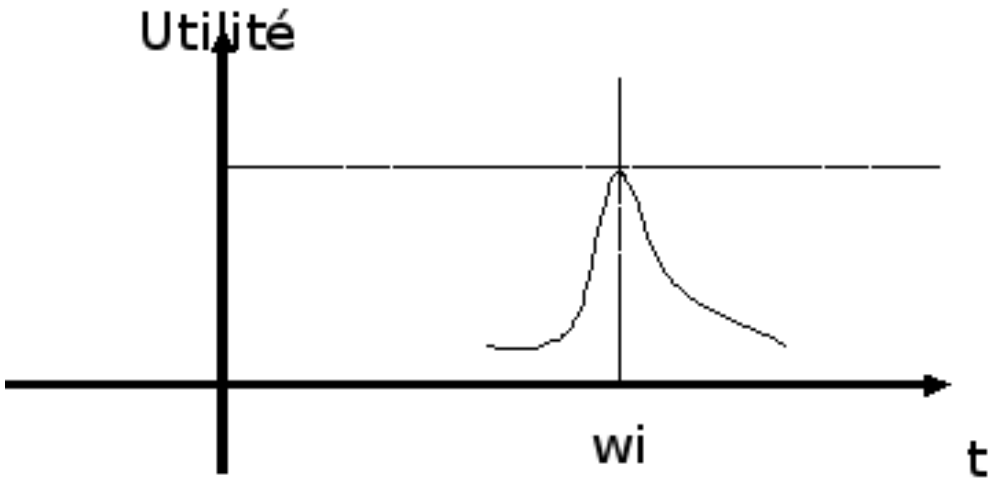
$\Rightarrow$ Meilleures précisions.
Applications directes:

- Traitement du signal
- Filtrage non linéaire
- Codage/décodage sans bruit
- Compression de données
- Cryptage

Utilité temporelle normale



Utilité temporelle dans le cas critique



Le caractère *temps-réel* d'un système découle de ses *spécifications*.
Les lois d'arrivées des entrées doivent être strictement définies et connues.
Les résultats fournis sont très souvent spécifiés de manières incomplètes:

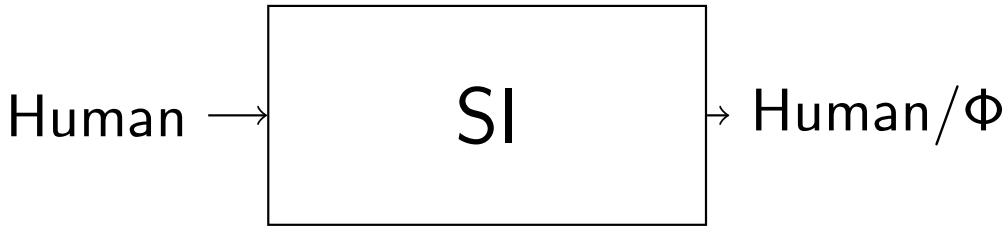
- interactions sur les données communes,
- utilités des résultats

Note: Deux systèmes ayant la même mise en œuvre peuvent être vus soit comme temps-réel, soit comme non temps-réel!

Note: le terme temps-réel est souvent employé à tord et à travers!
Ex: systèmes simples pour lesquels la contrainte temps-réel peut être négligée.
Ex: messagerie electronique
La réciproque est aussi vraie:
par abus de langage, un système fondé sur des techniques courantes en temps-réel sera souvent dit “temps-réel”.
Ex: systèmes multi-utilisateurs

Plan

Systèmes de transmission

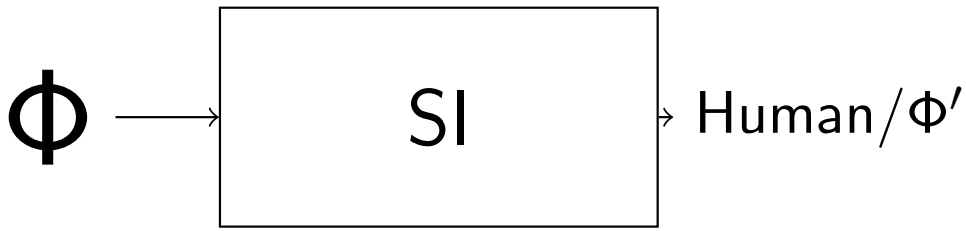


Peut être temps-réel:

- Débits ou temps de réponse
- Fiabilité
- Disponibilité

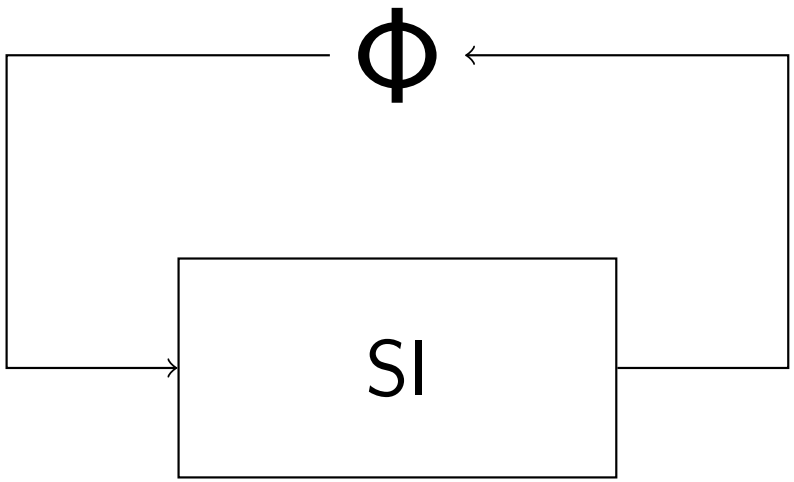
Note: un être humain peut toujours attendre un peu (non temps réel).

Systèmes d'acquisition



- Le système physique Φ impose le rythme d'acquisition
- Les entrées se font en temps-réel

Systèmes de contrôle/commande



Le système physique Φ impose ses lois de commande

Les entrées et les sorties se font en temps-réel

Le système est en boucle fermée

Systèmes de Contrôle-Commande multifonctions

- Examples :
  - Asservissements
  - Alarmes
  - Protections
  - Dialogues opérateur
  - Communications intersystèmes
- $\Rightarrow$ plusieurs flots de contrôles (de données)
  - Systèmes multitâches ou multithreads
  - Un seul système global à prédire
- $\Rightarrow$ composition des flots multiples en un sel
  - Produit synchronisé d'automates (dernier cours)
  - Ordonnancement (statique ou dynamique)

NB: modulaire n'implique pas compositionel!

Systèmes de Contrôle-Commande multifonctions

- Examples :
  - Asservissements
  - Alarmes
  - Protections
  - Dialogues opérateur
  - Communications intersystèmes
- $\Rightarrow$ plusieurs flots de contrôles (de données)
  - Systèmes multitâches ou multithreads
  - Un seul système global à prédire
- $\Rightarrow$ composition des flots multiples en un sel
  - Produit synchronisé d'automates (dernier cours)
  - Ordonnancement (statique ou dynamique)

NB: modulaire n'implique pas compositionel!

Systèmes de Contrôle-Commande multifonctions

- Examples :
  - Asservissements
  - Alarmes
  - Protections
  - Dialogues opérateur
  - Communications intersystèmes
- $\Rightarrow$ plusieurs flots de contrôles (de données)
  - Systèmes multitâches ou multithreads
  - Un seul système global à prédire
- $\Rightarrow$ composition des flots multiples en un sel
  - Produit synchronisé d'automates (dernier cours)
  - Ordonnancement (statique ou dynamique)

NB: modulaire n'implique pas compositionel!

Systèmes de Contrôle-Commande multifonctions

- Examples :
  - Asservissements
  - Alarmes
  - Protections
  - Dialogues opérateur
  - Communications intersystèmes
- $\Rightarrow$ plusieurs flots de contrôles (de données)
  - Systèmes multitâches ou multithreads
  - Un seul système global à prédire
- $\Rightarrow$ composition des flots multiples en un sel
  - Produit synchronisé d'automates (dernier cours)
  - Ordonnancement (statique ou dynamique)

NB: modulaire n'implique pas compositionel!

Systèmes de Contrôle-Commande multifonctions

- Examples :
  - Asservissements
  - Alarmes
  - Protections
  - Dialogues opérateur
  - Communications intersystèmes
- $\Rightarrow$ plusieurs flots de contrôles (de données)
  - Systèmes multitâches ou multithreads
  - Un seul système global à prédire
- $\Rightarrow$ composition des flots multiples en un sel
  - Produit synchronisé d'automates (dernier cours)
  - Ordonnancement (statique ou dynamique)

NB: modulaire n'implique pas compositionel!

La notion de flot de contrôle multiple est liée:

- aux acquisitions non synchrones (lois $\neq$)
- aux différentes courbes d'utilités

Critères de priorités:

- utilité directe ou indirecte,
- importance de la fonction.

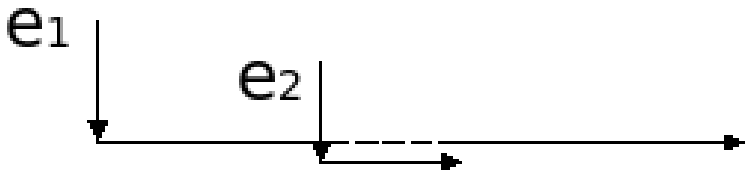
Conséquences:

- Préemption (interruption) de traitements long au profit de traitement cours,
- Entrelacement possible de plusieurs suites d'instructions (problèmes de cohérence possible),
- ou bien, éviter les arrivées asynchrones (difficile en pratique).

Plan

Ordonnancement et systèmes multitâches

Gérer les différentes échelles de temps: la préemption



Avantages

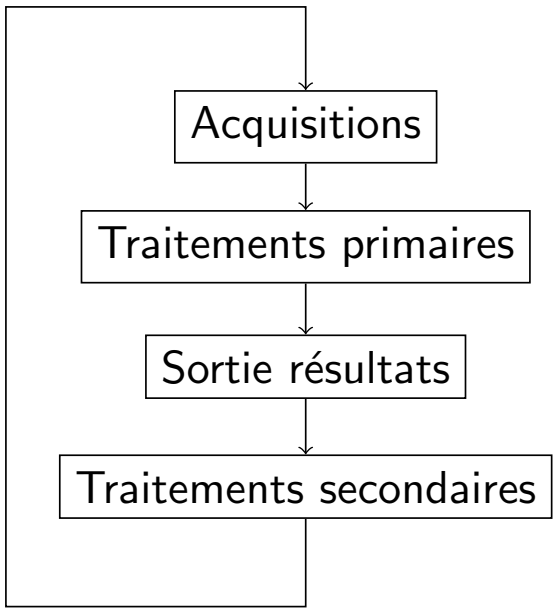
- Grande souplesse
- Facile
- Optimisation possible

Inconvénients

- Cohérence des données (entrelacement)
- Pertinence (violation d'une contrainte essentielle),
- Partage de ressources (dead-lock, famine)

Il n'existe pas d'algorithme d'ordonnancement "magique"

La programmation en boucle



On évite le multitâche en adoptant une programmation en boucle:

- flôt de contrôle statique
- pas de preemption.

Avantages "très simple", garanties par le temps de cycle.

Inconvénients : rigide, pas optimal, peut performant

Composition parallèle de programme

$P = P_1 * P_2$
Connaissant P_1 et P_2, quet peut-on dire de P?
Caractériser les interactions, explicites ou implicites :

- Cas asynchrone (produit possible ou pas),
- cas synchrone : produit synchronisé d'automates

La programmation en boucle permet une composition rigide et simple, mais ni facile ni efficace.
Une préemption peut introduire une désynchronisation.

Plan

Systèmes critiques

Confiance fondée sur quelques grands principes:

1. Robustesse, tolérance aux fautes
   (détection et confinement d'anomalies, modes dégradés, etc...)
2. Maîtrise des mécanismes mis en œuvre
   (ex: problèmes des logiciels à interruption)
3. Déterminisme comportemental
   (reproductibilité des tests et essais: les mêmes causes impliquent les mêmes effets)
4. Application du principe de diversification
   (prévenir les pannes de cause commune, ex: redondance ou diversité fonctionnelle, etc..)
5. Défense en profondeur

Reproductibilité et tests

- Les défauts logiciels
  - systématiques
- Les défaillances dues aux logiciels
  - systématiques (e.g. division par zéro)
  - non systématiques (e.g. synchronisation et non cohérence des données)
- difficulté majeure pour la sûreté de fonctionnement
- Les approches statistiques sont inadaptées pour le logiciel
  - dès que la probabilité de défaillance admissible du système est très basse
  - le logiciel n'a pas de fiabilité quantifiable
- Objectifs pour la sûreté de fonctionnement des systèmes à logiciel prépondérant
  - comportements du systèmes prédictibles
  - tests du systèmes reproductibles
- $\longrightarrow$ Construction du déterminisme

Tests et déterminisme

- Système non déterministe (ex: multitâche conventionnel): une petite variation des entrées (incontournable) entraîne une grande variation du comportement
- $\Rightarrow$ tests successifs peuvent donner des résultats différents
- tests et système final peuvent donner des résultats différents
  - Le test a peu de valeur pour la validation du système
- Il faut agir en amont pour assurer structurellement le déterminisme

- Activités à exécuter sur des calculateurs:
  - Acquérir les données décrivant le procédé et l'environnement (entrées)
  - Traiter ces données (algorithmes)
  - Commander des actionneurs (sorties)
- Contraintes temporelles spécifiées (explicites ou non):
  - Rythmes: $T_{start} = T_0, T_1, T_2 \ldots$
  - Intervalles temporels de validité pour les activités élémentaires: $T_{start} < AE < T_{end}$
  - Dépendences entre les activités (relations d'ordres, sections critiques)
- Coordination de toutes les activités:
  - Cohérence temporelle: synchronisations sur le temps physique (temps réel);
  - Branchements conditionnels : "synchronisations logiques"
- Problèmes:
  - De quelques centaines à quelques milliers d'activités élémentaires à coordonner
- $\Rightarrow$ *La coordination déterministe de toutes les activités est difficile*

Choix d'implantation

Modèle d'exécution séquentiel (découpage statique) (ex: programmation en boucle)

Avantage : déterministe

Désavantages :

- contraintes très fortes sur la réalisation (réalistes?)
- ne profite pas des architectures multi processeur.

Modéle d'exécution parallèle (découpage dynamique, préemptions)

Problème: vérifications analytiques possibles?

Le déterminisme idéal

- Déterminisme logique et temporels sont indissociables
  - unicité et invariance du comportement dynamique
  - asynchronismes du système maîtrisés
- Impact du déterminisme
  - comportement dynamique du système prédictible et invariant:
    - reproductibilité du comportement dynamique
  - tests (vérification/validation pour la sûreté) :
    - exhaustivité des comportements atteignable (recherche du pire des cas)
  - indépendance de l'implantation
  - simulation exacte sans la machine cible
- Empêcher les propagations d'erreurs et maîtriser l'impact des anomalies
  - mécanismes déterministes de détection et de confinement d'anomalies
- *L'impact d'une anomalie de fonctionnement doit (devrait) être déterministe*