Ordonnancement temps réel monoprocesseur

Frank Singhoff

Bureau C-203

Université de Brest, France

Laboratoire Lab-STICC UMR CNRS 6285

singhoff@univ-brest.fr

Sommaire

Ordonnancement, définitions (1)

- **Objectifs** : prendre en compte les besoins d'urgence, d'importance des applications temps réel.

- **Eléments de taxinomie** :

  - Algorithmes hors ligne/en ligne : moment où sont effectués les choix d'allocation.

  - Priorités statiques/dynamiques : les priorités changent elles ? Algorithmes statiques/dynamiques.

  - Algorithmes préemptifs ou non : tâches interruptibles par d'autres ? non préemptif =
    1. Exclusion mutuelle des ressources aisée.
    2. Surcoût de l'ordonnanceur moins élevé.
    3. Efficacité moindre.

Ordonnancement, définitions (2)

- **Propriétés recherchées :**

1. **Faisabilité/ordonnançabilité :** est il possible d'exhiber un test de faisabilité ?
   - Condition permettant de décider hors ligne du respect des contraintes des tâches.
   - Exemple : pire temps de réponse des tâches.

2. **Optimalité :** critère de comparaison des algorithmes (un algorithme est dit optimal s'il est capable de trouver un ordonnancement pour tout ensemble faisable de tâches).

3. **Complexité :** les tests de faisabilité sont ils polynômiaux ? exponentiels ? passage à l'échelle ?

4. **Facilité de mise en œuvre :** l'ordonnanceur est-il facile à implanter dans un système d'exploitation ?

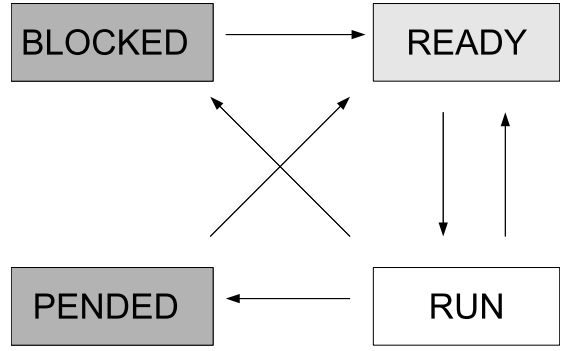
Vérification/validation d'un système

- **Exemple de processus de vérification d'un système:**

1. On définit l'architecture matérielle et l'environnement d'exécution: la capacité mémoire, le processeur et sa puissance de calcul, le système d'exploitation (et donc l'algorithme d'ordonnancement des tâches).
2. On réalise le code fonctionnel (ex: fonction C d'un thread POSIX).
3. On conçoit l'architecture logicielle: comment associer le code fonctionnel et le matériel. Conduit à modéliser les tâches du système ainsi que leurs contraintes et caractéristiques temporelles.
4. On valide la faisabilité de l'architecture logicielle sur l'architecture matérielle, c-a-d, entre autre la faisabilité du jeu de tâches.

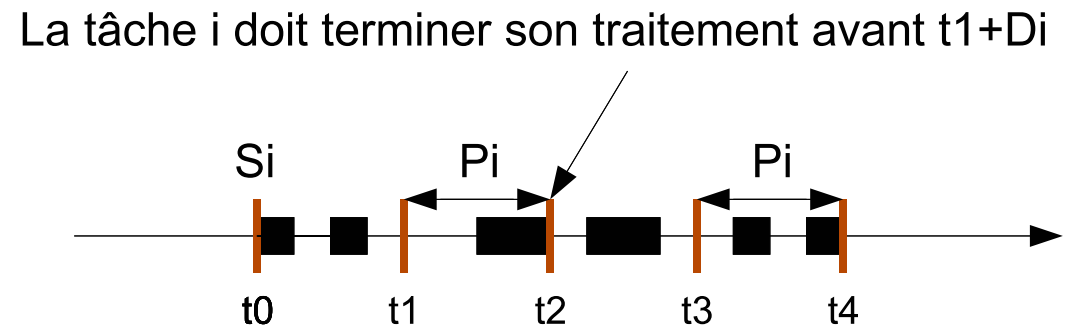
5. Eventuellement, on revient à 1, 2 ou 3. Cycle de conception/vérification.

Notion de tâche (1)



- **Processeur** = unique ressource partageable dans un système temps réel $\Longrightarrow$ exécutif temps réel.
- **Tâche** : suite d'instructions + données + contexte d'exécution (état).
- **Famille de tâches** :
  - Tâches dépendantes ou non. Tâches importantes, urgentes.
  - Tâches répétitives : activations successives (tâches périodiques ou sporadiques) $\Longrightarrow$ **fonctions critiques**.
  - Tâches non répétitives/apériodiques : une seule activation $\Longrightarrow$ **fonctions non critiques**.

Notion de tâche (2)



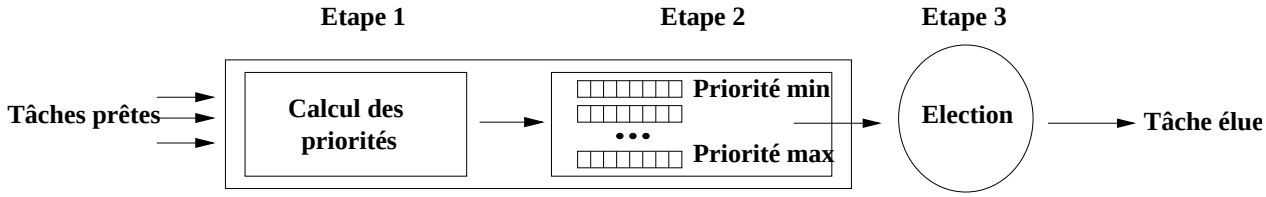
- **Paramètres définissant une tâche i :**
  - Arrivée de la tâche dans le système : S_i.
  - Pire temps d'exécution d'une activation : C_i (capacité).
  - Période d'activation : P_i.
  - Délai critique : D_i (relatif à P_i si tâche périodique, à S_i si tâche apériodique).
  - Date d'exécution au plus tôt : R_i.

Notion de tâche (3)

- Un modèle simplifié de tâches : **le modèle de tâches périodiques synchrones à échéance sur requête [LIU 73]**.

- **Caractéristiques :**
  - Tâches périodiques.
  - Tâches indépendantes.
  - avec $\forall i : S_i = 0 \implies$ instant critique (pire cas).
  - avec $\forall i : P_i = D_i \implies$ tâches à échéance sur requête.

Ordonnanceur : structure, fonctions (1)

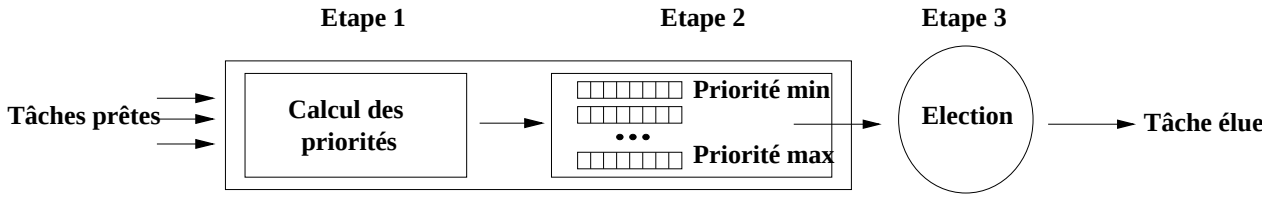


- **Trois fonctions principales** :

(1) **Calcul de priorité :**

- En ligne ou hors ligne.
- Période : Rate Monotonic (RM).
- Echéance : Deadline Monotonic (DM), Earliest Deadline First (EDF).
- Laxité : Least Laxity First (LLF). (laxité à l'instant t : $L_i(t) = D_i(t) -$ reliquat de C_i à exécuter).
- Etc.

Ordonnanceur : structure, fonctions (2)



(2) **Gestion de la/des files d'attente :**

- Une file par priorité.
- Application d'une politique (FIFO, Round-Robin, ...).
- Exemple : POSIX 1003.1b, Chorus, Solaris, etc ...

(3) **Election :**

- Election de la tâche en tête de file.
- HPF : plus haute priorité d'abord.
- EDF : plus courte échéance d'abord.
- LLF : plus petite laxité d'abord.

Sommaire

1. Introduction et concepts de base.
2. Algorithmes classiques pour le temps réel.
3. Un peu de pratique : le standard POSIX 1003.
4. Prise en compte de dépendances.
5. Outils de vérification.
6. Résumé.
7. Références.

Algorithmes classiques

1. Algorithme à priorités fixes, avec affectation des priorités selon Rate Monotonic (RM, RMS, RMA).
2. Algorithme à priorités dynamiques, Earliest Deadline First (EDF).

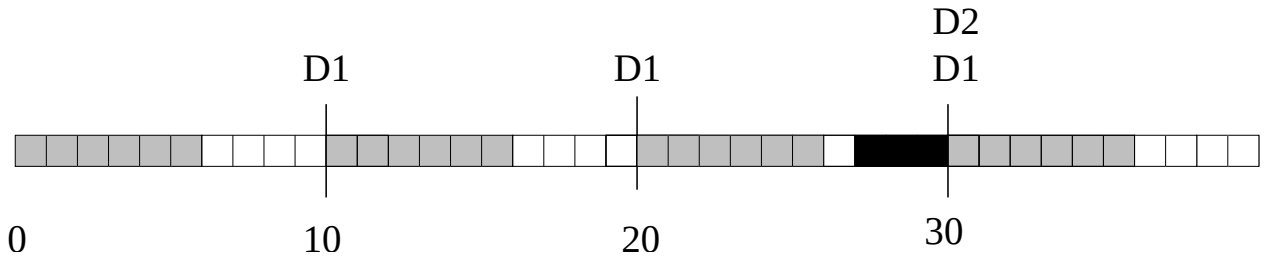
Ordonnancement à priorité fixe (1)

- **Caractéristiques :**
  - Priorités fixes $\Longrightarrow$ analyse hors ligne $\Longrightarrow$ applications statiques et critiques.
  - Complexité faible et mise en œuvre facile dans un système d'exploitation.
- **Fonctionnement :**
  1. Affectation hors ligne des priorités.
  2. Election de la tâche de plus forte priorité.
- **Affectation des priorités selon Rate Monotonic :**
  - Algorithme optimal dans la classe des algorithmes à priorité fixe.
  - Tâches périodiques uniquement.
  - Priorité = inverse de la période.

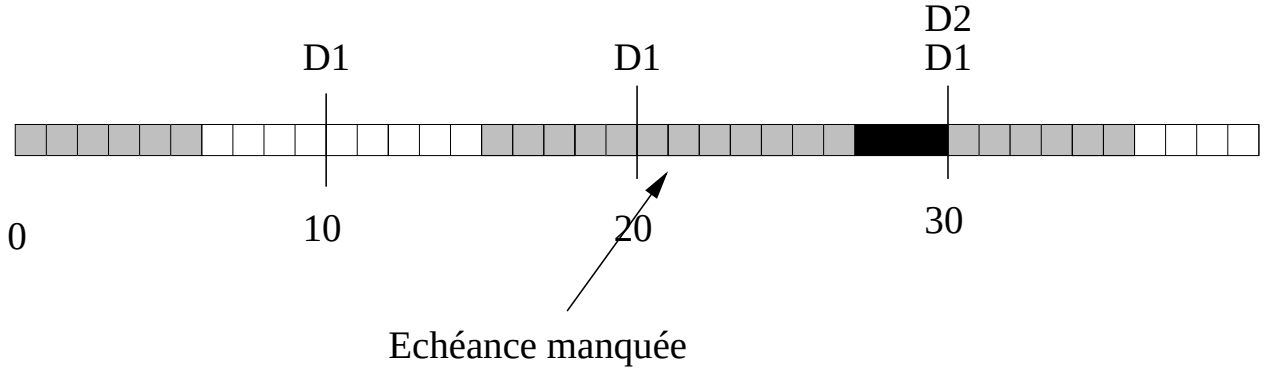
Ordonnancement à priorité fixe (2)

- Cas préemptif :

T1 : C1=6 ; P1=10 (gris)
T2 : C2=9 ; P2=30 (blanc)
Noir=libre



- Cas non préemptif :



Ordonnancement à priorité fixe (3)

- **Faisabilité/ordonnançabilité :**

1. **Période d'étude** = $[0, PPCM(P_i)]$. Solution exacte. Toute affectation de priorité. Préemptif ou non.

2. **Taux d'utilisation** (préemptif et RM seulement) :

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

   Condition suffisante mais non nécessaire : solution pessimiste donc.

3. **Temps de réponse** $\Longrightarrow$ délai entre l'activation d'un tâche et sa terminaison. Solution parfois exacte (selon les modèles de tâches). Toute affectation de priorité. Préemptif ou non.

Ordonnancement à priorité fixe (4)

- **Calcul du pire temps de réponse :**
  - Hypothèses : cas préemptif.
  - Principe : pour une tâche i, on cherche à évaluer, au pire cas, son temps d'exécution + son temps d'attente lorsque des tâches plus prioritaires s'exécutent. Où encore :

$$r_i = C_i + \sum_{j \in hp(i)} I_j$$

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

  - Où $hp(i)$ est l'ensemble des tâches de plus forte priorité que i ; $\lceil x \rceil$ est l'entier directement plus grand que x.

Ordonnancement à priorité fixe (5)

- **Technique de calcul** : on évalue de façon itérative w_i^n par :

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

- On démarre avec $w_i^0 = C_i$.
- Conditions d'arrêt :
  - Echec si $w_i^n > P_i$.
  - Réussite si $w_i^{n+1} = w_i^n$.

Ordonnancement à priorité fixe (6)

- **Exemple :** P1=7 ; C1=3; P2=12 ; C2=2 ; P3=20 ; C3=5

  - $w_1^0 = 3 \Longrightarrow r_1 = 3$
  - $w_2^0 = 2$
  - $w_2^1 = 2 + \lceil \frac{2}{7} \rceil 3 = 5$
  - $w_2^2 = 2 + \lceil \frac{5}{7} \rceil 3 = 5 \Longrightarrow r_2 = 5$
  - $w_3^0 = 5$
  - $w_3^1 = 5 + \lceil \frac{5}{7} \rceil 3 + \lceil \frac{5}{12} \rceil 2 = 10$
  - $w_3^2 = 5 + \lceil \frac{10}{7} \rceil 3 + \lceil \frac{10}{12} \rceil 2 = 13$
  - $w_3^3 = 5 + \lceil \frac{13}{7} \rceil 3 + \lceil \frac{13}{12} \rceil 2 = 15$
  - $w_3^4 = 5 + \lceil \frac{15}{7} \rceil 3 + \lceil \frac{15}{12} \rceil 2 = 18$
  - $w_3^5 = 5 + \lceil \frac{18}{7} \rceil 3 + \lceil \frac{18}{12} \rceil 2 = 18 \Longrightarrow r_3 = 18$

Ordonnancement à priorité fixe (7)

- Comment faire cohabiter des tâches apériodiques dans un système ordonnancé par priorité fixe avec Rate Monotonic :

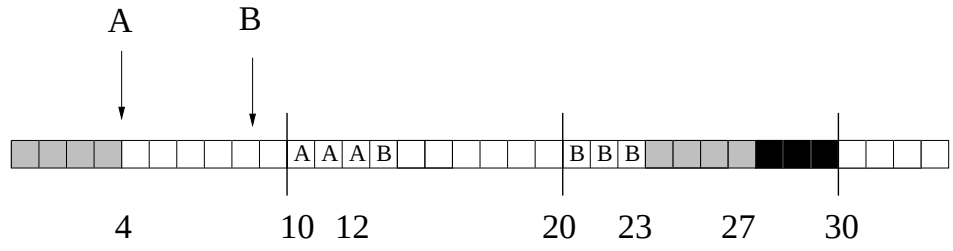
1. Les tâches apériodiques ne sont pas urgentes $\Longrightarrow$ priorités plus faible que les tâches périodiques.
2. Les tâches apériodiques sont urgentes $\Longrightarrow$ utilisation de serveur de tâches apériodiques.

Ordonnancement à priorité fixe (8)

- **Serveur de tâches apériodiques** : tâche périodique dédiée à l'exécution des tâches apériodiques.

T1 : C1=4 ; P1=20 (gris)
T2 : C2=12 ; P2=30 (blanc)
S : CS=4 ; PS=10;
A arrive en t=4 ; CA=3
B arrive en t=9 ; CB=4

Noir=libre



- **Serveur par scrutation :** exécution des tâches apériodiques arrivées avant le début de l'activation du serveur ; ne consomme pas de temps processeur si pas de tâche apériodique. Temps de scrutation considéré négligeable. Simple mais ressource gaspillée.
- **Autres serveurs :** serveur sporadique, différé, ...

L’algorithme EDF (1)

- **Caractérisques :**
  - Algorithme à priorité dynamique $\Longrightarrow$ mieux adapté que priorité fixe aux applications dynamiques.
  - Supporte les tâches périodiques et les tâches apériodiques.
  - Algorithme optimal : utilise jusqu’à 100 pourcents de la ressource processeur.
  - Mise en œuvre difficile dans un système d’exploitation.
  - Instable en sur-charge : moins déterministe que priorité fixe.

L'algorithme EDF (2)

- **Fonctionnement :**

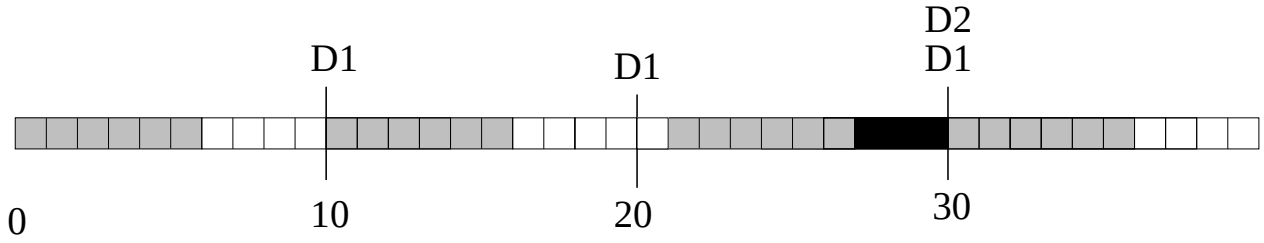
1. Calcul de la priorité $\Longrightarrow$ calcul d'une échéance . Soit $D_i(t)$, l'échéance à l'instant t et D_i, le délai critique de la tâche i :
   - Tâche apériodique : $D_i(t)$ = D_i + S_i.
   - Tâche périodique : $D_i(t)$ = date de début de l'activation courante à l'instant t + D_i.

2. Election : élection de la plus courte échéance d'abord.

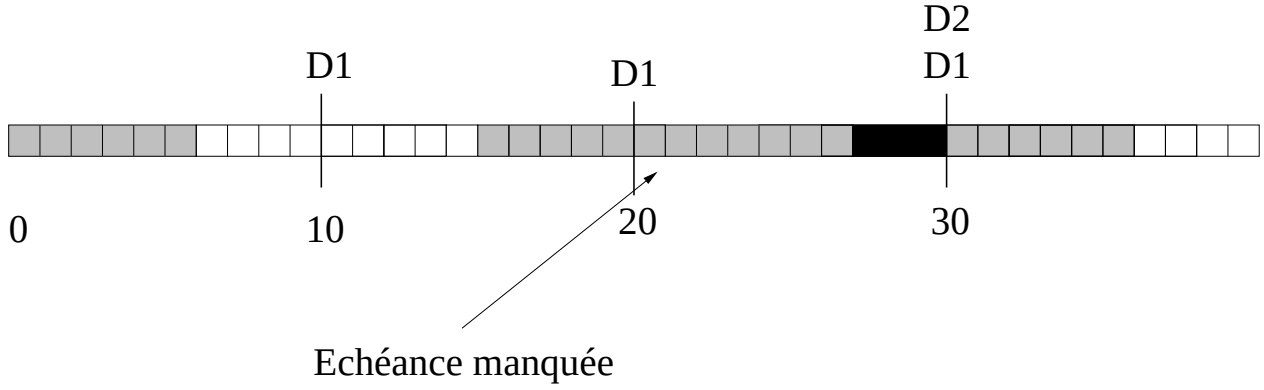
L'algorithme EDF (3)

- Cas préemptif :

T1 : C1=6 ; P1=10 (gris)
T2 : C2=9 ; P2=30 (blanc)
Noir = libre



- Cas non préemptif :



L'algorithme EDF (4)

- **Faisabilité/Ordonnançabilité :**

1. **Période d'étude** : idem priorité fixe (tâches périodiques).
2. **Taux d'utilisation**, cas préemptif, tâches périodiques indépendantes et synchrones:
   - Condition nécessaire et suffisante si $\forall\, i : D_i = P_i$:

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

     (uniquement nécessaire si $\exists\, i \;:\; D_i < P_i$)
   - Condition suffisante si $\exists\, i \;:\; D_i < P_i$:

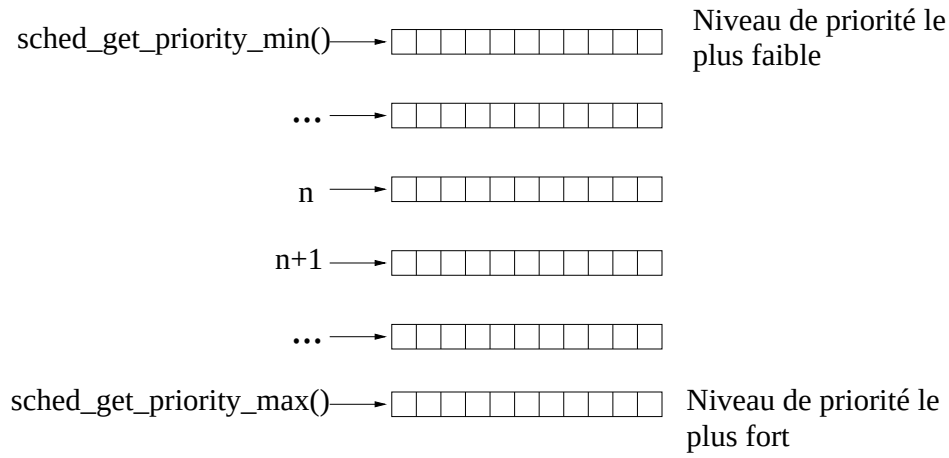
$$U = \sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1$$

3. **Temps de réponse :** complexité importante.

Sommaire

1. Introduction et concepts de base.
2. Algorithmes classiques pour le temps réel.
3. Un peu de pratique : le standard POSIX 1003.
4. Prise en compte de dépendances.
5. Outils de vérification.
6. Résumé.
7. Références.

La norme POSIX 1003.1b (1)



- POSIX 1003.1b [GAL 95] = extensions temps réel du standard ISO/ANSI POSIX définissant une interface portable de systèmes d'exploitation.

- **Modèle d'ordonnancement :**
  - Priorités fixes, préemptif $\Longrightarrow$ RM facile.
  - Une file d'attente par priorité + politiques de gestion de la file ($SCHED\_FIFO$, $SCHED\_RR$, $SCHED\_OTHERS$, ...).

La norme POSIX 1003.1b (2)

- **Gestion des files d'attente POSIX :** élection de la tâche en tête de file d'attente de plus haute priorité.

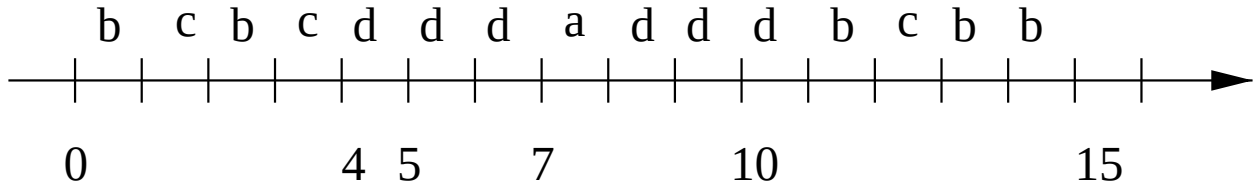
- **Principales politiques proposées :**

1. $SCHED\_FIFO$: la tâche quitte la tête de file si :
   - Terminaison de la tâche.
   - Blocage de la tâche (E/S, attente d'un délai) => remise en queue.
   - Libération explicite => remise en queue.
2. $SCHED\_RR$: idem $SCHED\_FIFO$ mais en plus, la tâche en tête de file est déplacée en queue après expiration d'un quantum ($round\ robin$).
3. $SCHED\_OTHERS$: fonctionnement non normalisé.

La norme POSIX 1003.1b (3)

- Exemple :

Tâches	C_i	S_i	Priorité	Politique
a	1	7	1	FIFO
b	5	0	4	RR
c	3	0	4	RR
d	6	4	2	FIFO



- Quantum SCHED_RR = 1 unité de temps.
- Niveau de plus forte priorité : 1.

La norme POSIX 1003.1b (4)

- Ordonnanceurs POSIX 1003.1b :

```
#define SCHED_OTHER     0
#define SCHED_FIFO      1
#define SCHED_RR        2
```

- Consultation des paramètres spécifiques à la mise en œuvre de POSIX 1003.1b :

```
int sched_get_priority_max(int policy);
int sched_get_priority_min(int policy);
int sched_rr_get_interval(pid_t pid,
        struct  timespec *tp);
```

- Libération volontaire du processeur :

```
int sched_yield(void);
```

La norme POSIX 1003.1b (5)

- Paramètre(s) ordonnanceur :

```
struct sched_param
  {
    int sched_priority;
    ...
  };
```

- Consultation ou modification de l'ordonnanceur :

```
int sched_setscheduler(pid_t pid, int policy,
        const struct sched_param *p);
int sched_getscheduler(pid_t pid);
```

- Consultation ou modification des paramètres d'ordonnancement :

```
int  sched_getparam(pid_t  pid,
        struct sched_param *p);
int   sched_setparam(pid_t   pid,
        const struct sched_param *p);
```

La norme POSIX 1003.1b (6)

- Initialisation : héritage par $fork()$, démarrage en section critique.

- Exemple : cf. tâches page 14.

```
struct sched_param parm;
int res=-1;
...
/* Tache T1 ; P1=10 */
parm.sched_priority=15;
res=sched_setscheduler(pid_T1,SCHED_FIFO,&parm);
if(res<0)
     perror("sched_setscheduler tache T1");

/* Tache T2 ; P2=30 */
parm.sched_priority=10;
res=sched_setscheduler(pid_T2,SCHED_FIFO,&parm);
if(res<0)
     perror("sched_setscheduler tache T2");
```

Sommaire

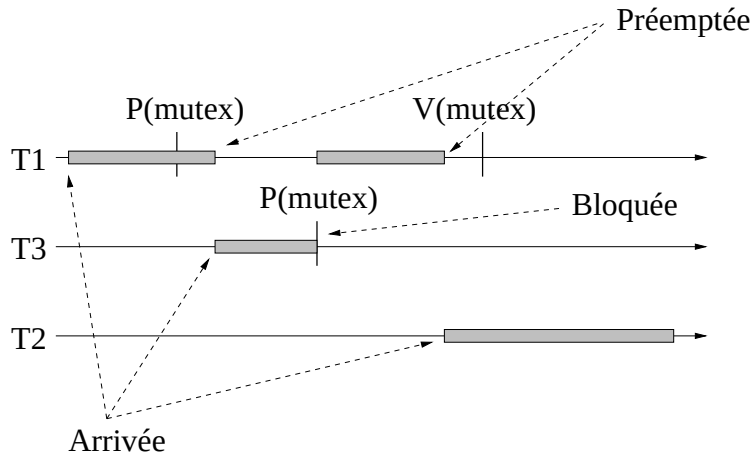
1. Introduction et concepts de base.
2. Algorithmes classiques pour le temps réel.
3. Un peu de pratique : le standard POSIX 1003.
4. Prise en compte de dépendances.
5. Outils de vérification.
6. Résumé.
7. Références.

Prise en compte des dépendances

- **Différentes dépendances:**

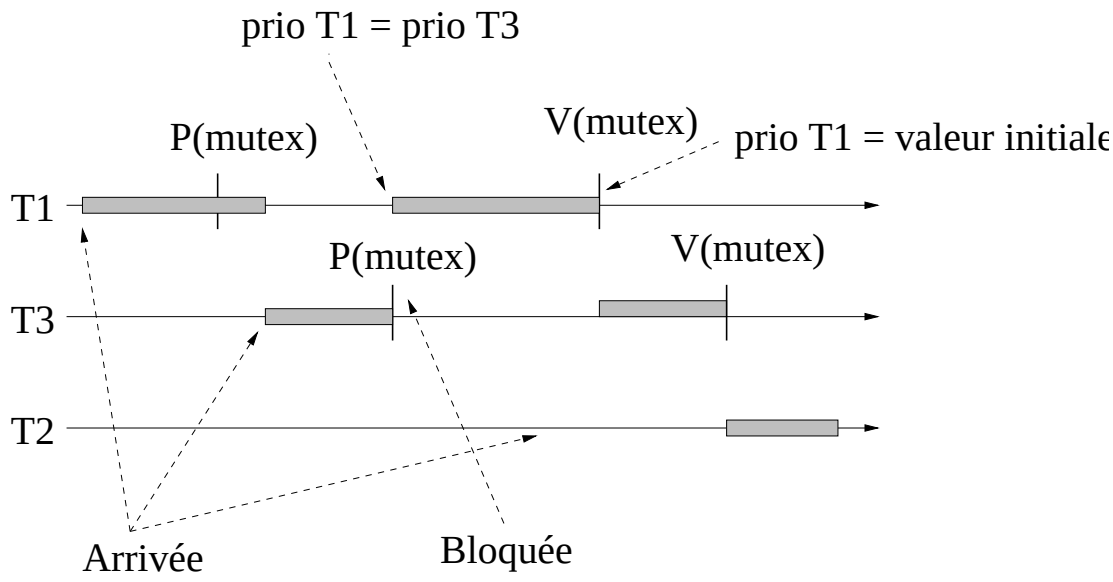
1. Partage de ressources.
2. Contraintes de précédence (ex : communications).

Partage de ressources (1)



- Accéder à une ressource c'est éventuellement devoir attendre qu'elle se libère = borne sur le temps de blocage (noté B_i).
- **Problèmes :**
  - Iinversion de priorités : comment réduire la durée de l'inversion de priorité ? $\Longrightarrow$ protocoles d'héritage de priorité.
  - Comment finement évaluer B_i ?
- Caractéristiques des protocoles : calcul de B_i, nombres de ressources accessibles, complexité, interblocage possible ou non, etc.

Partage de ressources (2)



- **Héritage simple (ou Priority Inheritance Protocol ou PIP) :**
  - Une tâche qui bloque une autre plus prioritaire qu'elle, exécute la section critique avec la priorité de la tâche bloquée.
  - Une seule ressource : sinon interblocage possible.
  - $B_i =$ somme des sections critiques des tâches moins prioritaires que i.

Partage de ressources (3)

- PIP ne peut pas être utilisé avec plusieurs ressources $\Longrightarrow$ interblocage. On implante généralement PCP (Priority Ceiling Protocol) [SHA 90] (ex. VxWorks).

- Deux variétés de PCP : OCPP et ICPP.

- **Exemple d'Immediate Ceiling Priority Protocol ou ICPP :**

  - Priorité plafond d'une ressource = priorité statique maximale de toutes les tâches qui utilisent la ressource.
  - Priorité dynamique d'une tâche = maximum (priorité statique de la tâche, priorité plafond de toutes les ressources allouées).
  - $B_i =$ plus grande section critique.

Partage de ressources (4)

- Soit n tâches périodiques synchrones à échéance sur requête, ordonnées de façon décroissantes selon leur priorité (avec $B_n = 0$ donc).
- Prise en compte du temps de blocage dans :
  - Critère d'ordonnançabilité RM :

$$\forall\, i, 1 \leq i \leq n : \sum_{k=1}^{i-1} \frac{C_k}{P_k} + \frac{C_i + B_i}{P_i} \leq i(2^{\frac{1}{i}} - 1)$$

  - Critère d'ordonnançabilité EDF/LLF :

$$\forall\, i, 1 \leq i \leq n : \sum_{k=1}^{i-1} \frac{C_k}{P_k} + \frac{C_i + B_i}{P_i} \leq 1$$

Partage de ressources (5)

- Prise en compte du temps de blocage B_i dans le calcul du temps de réponse d'un ensemble de tâches périodiques synchrones à échéances sur requêtes, ordonnancées priorité fixe préemptif :

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

Où $hp(i)$ est l'ensemble des tâches de plus forte priorité que i.

- Résolution par la méthode itérative :

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

Attention : avec B_i, le calcul du temps de réponse devient une condition suffisante mais non nécessaire. C'est une solution pessimiste.

Contraintes de précédence (1)

- **Principales approches :**

1. **Conditions initiales** (paramètre S_i). Décaler les réveils selon les contraintes de précédence. Tests d'ordonnançabilité spécifiques.
2. **Affectation des priorités** (Chetto/Blazewicz [BLA 76, CHE 90]).
3. **Modification des délais critiques** (Chetto/Blazewicz).
4. **Utilisation des paramètres "Jitter" et/ou "offset"** [TIN 94, TIN 92]. Calcul de pire temps de réponse
5. **Heuristique d'ordonnancement** (ex : Xu et Parnas [XU 90]).

Contraintes de précédence (2)

- **Principe de Blazewicz [BLA 76] et Chetto et al [CHE 90] :** rendre les tâches indépendantes en modifiant leurs paramètres.

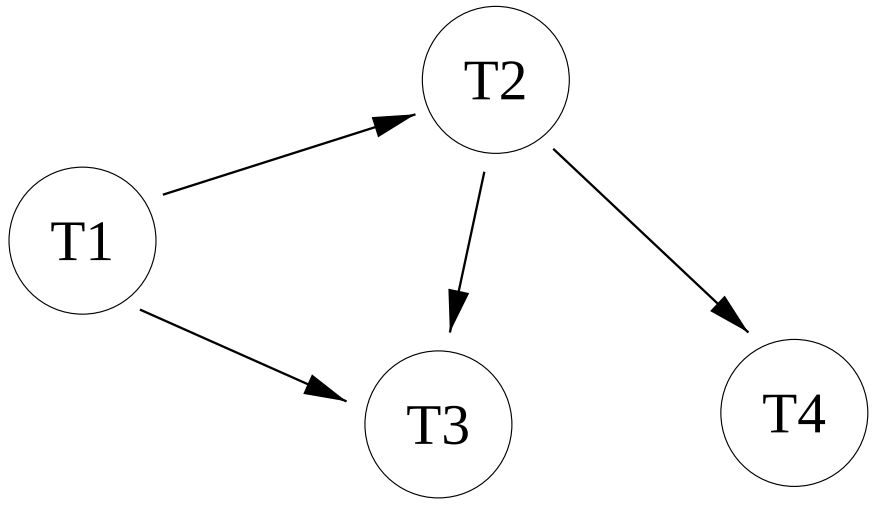
- **Hypothèses :** Tâches soit apériodiques, soit périodiques de même période.

- **Technique :**

1. Modification pour RM :
   - $\forall\, i, j \mid i \prec j \;:\; priorite_i > priorite_j$

2. Modification pour EDF :
   - $D_i^* = min(D_i, min(\forall\, j \mid i \prec j \;:\; D_j^* - C_j))$.

Contraintes de précédence (3)

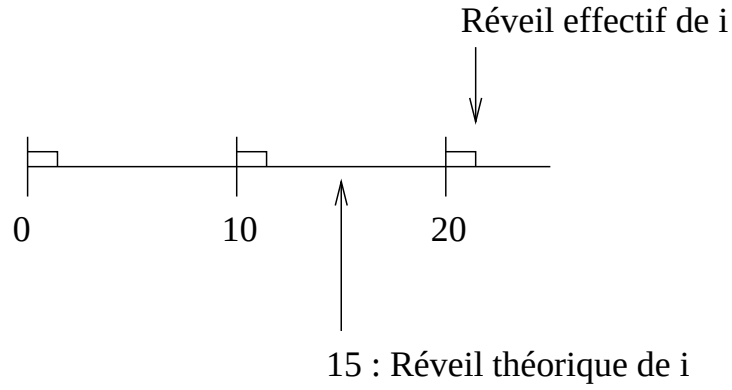


	C_i	D_i	D_i^*
T_4	2	14	14
T_3	1	8	8
T_2	2	10	7
T_1	1	5	5

- **Exemple : EDF + tâches apériodiques.**
  - $D_4^* = 14$;
  - $D_3^* = 8$;
  - $D_2^* = min(D_2, D_3^* - C_3, D_4^* - C_4) = min(10, 8-1, 14-2) = 7$;
  - $D_1^* = min(D_1, D_2^* - C_2, D_3^* - C_3) = min(5, 7-2, 8-1) = 5$;

Contraintes de précédence (4)

- Utilisation du **Jitter**. Exemple historique $\Longrightarrow$ le timer d'un système est modélisé comme une tâche périodique avec $P_{timer} = 10\ ms, C_{timer} = 3\ ms$.
- On souhaite réveiller une tâche i à l'instant $t = 15\ ms$.



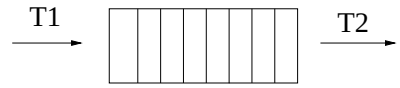
La date effective de réveil de la tâche i sera $23ms$. Sa gigue est de $J_i = 8\ ms$.

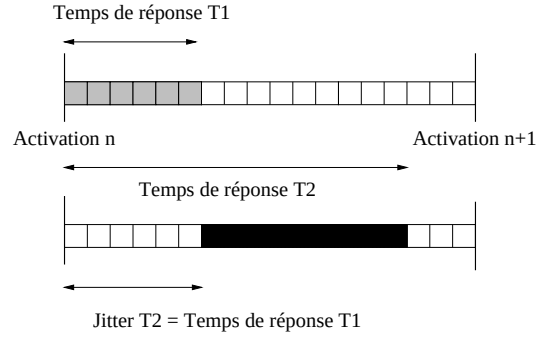
- Temps de réponse = $r_i = w_i + J_i$, avec :

$$w_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i + J_j}{P_j} \right\rceil C_j$$

Contraintes de précédence (5)

T1 : C1=6 ; P1=18 (gris)
T2 : C2=9 ; P2=18 (noir)





- **Exemple du producteur/consommateur :**
  - T1 et T2 sont activées toutes les 18 unités de temps.
  - T1 lit un capteur et transmet la valeur vers T2 qui doit l'afficher à l'écran.
  - T2 doit être activée sur terminaison de T1.
  - Quel est le temps de réponse de T2 ?

Sommaire

1. Introduction et concepts de base.
2. Algorithmes classiques pour le temps réel.
3. Un peu de pratique : le standard POSIX 1003.
4. Prise en compte de dépendances.
5. Outils de vérification.
6. Résumé.
7. Références.

Outils de vérification (1)

- **Un outil d'analyse de l'ordonnancement temps réel doit offrir :**
  - Un moyen pour décrire le système à vérifier (architecture et comportement).
  - Des moyens d'analyse à proprement dit, qui peuvent être basés sur :
    1. **Méthodes algébriques/analytiques:** tests de faisabilité.
    2. **Le Model-checking:** le système est décrit grâce à un modèle avec un langage formel, afin d'énumérer exhaustivement l'ensemble des états potentiels du modèle.
    3. **La simulation:** calcul de chronogrammes puis analyse (vérification généralement partielle).

Outils de vérification (2)

	Algèbrique/analytiqye	Model checking	Simulation
Preuve	oui	oui	non
Système de grande taille	oui	parfois	parfois
Tâches/ordonnanceurs spécifiques	non	oui	oui
Facile à employer	oui	non	parfois

- **Quel type d'outil doit on employer ?** Tous ...

ils sont complémentaires.

Outils de vérification (3)

- **Exemples d'outils commerciaux/open-source :**

  - MAST, Université de Cantabria, `http://mast.unican.es/`
  - Rapid-RMA, Tri-Pacific Software Inc, `http://www.tripac.com/`
  - Times, Université de Uppsala, `http://www.timestool.com/`
  - Cheddar, Université de Brest, `http://beru.univ-brest.fr/~singhoff/chedda`
  - ...

Outils de vérification (4)

- **Cheddar** = outil pédagogique développé par l'Université de Bretagne Occidentale/Lab-STICC.
- **Modèle d'architecture :**
  - Support pour divers langages d'architecture: AADL, UML/Marte, ...
  - Propose un modèle d'architecture propriétaire.
  - Inter-opérabilité avec différents outils de modélisation (Stood, TOPCASED, IBM Rational Software Architect, POOA-visio).
- **Sujets de TD + corrections :** http://beru.univ-brest.fr/~singhoff/cheddar/contribs/educational/ubo

Outils de vérification (5)

- **Architecture and Analysis Design Language (AADL) :**
  - Norme internationale publiée par la SAE (Society of Automotive Engineers) sous le standard AS-5506.
  - Version 1.0 publiée en 2004, puis version 2 en 2009. http://aadl.info rassemble les informations utiles sur AADL.
  - Langage de conception/modélisation adapté aux systèmes répartis embarqués temps réel :
    1. Concurrence.
    2. Modélisation de l'architecture logicielle et matérielle (quantifier les ressources).
    3. Expression contraintes temporelles.

Outils de vérification (6)

- **Composant AADL :**
  - **Définition d'un composant :** représentation d'une entité logicielle ou matérielle, réutilisable/paquetage. Un type/interface + une ou plusieurs implantations.
  - **Interaction entre composants :** features (interface) + connexions.
  - **Propriétés de composant :** attributs qui indique toutes informations nécessaires à sa mise en oeuvre ou à son analyse.
  - Un composant peut avoir des sous-composants.

$\Longrightarrow$ Modèle d'architecture AADL = hiérarchie/arborescence de composants.

Outils de vérification (7)

- **Déclaration d'un composant :**
  - **Type du composant :** identificateur, catégorie, propriétés et features.
  - **Implantation du composant :** structure interne (sous-composants, propriétés, ...).
  - **Catégorie du composant :** modélise les abstractions présentes dans un système temps réel (sémantique, comportement du composant).
  - **Trois familles de catégories :** matériels (composants constituant la plate-forme d'exécution), logiciels (composants constituant le logiciel à réaliser), systèmes (architecture globale du système à réaliser).

Outils de vérification (8)

- **Catégories de composants logiciels :**
  - **thread :** fil d'exécution (flot de contrôle qui exécute un programme) => tâche Ada/VxWorks, thread POSIX/Java, ...
  - **data :** structure de données implantée dans le langage cible => struct C, class C++/Java, record Ada, ...
  - **process :** modélise un espace mémoire (protection mémoire). Un process doit contenir au moins un thread.
  - **subprogram :** modélise un programme exécutable séquentiellement (fonction C, méthode Java, sous-programme Ada). Est associé à un code source.
  - **thread group :** modélise la notion de hiérarchie entre threads

Outils de vérification (9)

- **Exemple de composants logiciels :**

```
thread receiver
end receiver;

thread implementation receiver.impl
end receiver.impl;

thread analyser ...
thread implementation analyser.impl ...

process processing
end processing;

process implementation processing.others
subcomponents
   receive : thread receiver.impl;
   analyse : thread analyser.impl;
end processing.others;
```

Outils de vérification (10)

- **Catégories de composants matériels :**
  - **processor :** abstraction du logiciel/matériel en charge de l'ordonnancement des threads. Un processor peut comporter plusieurs virtual processors.
  - **memory :** modélise toute entité de stockage physique de données (disque dur, mémoire vive, etc).
  - **device :** composant qui interagit avec l'environnement. On ignore sa structure interne (thread, data, ...). Ex : capteur, actionneur.
  - **bus :** entité permettant l'échange de donnée/contrôle entre device, memory ou processor (ex : réseau de communication, bus, etc).

- **Exemple :**

```
device antenna
end antenna;
```

```
processor leon2;
end leon2;
```

Outils de vérification (11)

- **Catégorie «system» :**
  - Permet de structurer un modèle d'architecture tout en manipulant les sous-composants indépendamment (system, process, processor, device, bus).
  - Spécifie le déploiement des composants logiciels sur les composants matériels.
  - Un composant système constitue la racine de l'arborescence de l'architecture.

Outils de vérification (12)

- **Exemple d'un système complet :**

```
thread implementation receiver.impl ...
process implementation processing.others ...
processor leon2  ...

system radar
end radar;

system implementation radar.simple
subcomponents
   main : process processing.others;
   cpu : processor leon2;
properties
   Actual_Processor_Binding =>
      reference cpu applies to main;
end radar.simple;
```

Outils de vérification (13)

- **Propriété :**
  - Attribut typé, associé à un ou plusieurs composants.
  - Propriété = Nom + type + liste des composants concernés.
  - Property sets. Property sets définis par la norme : $AADL\_Properties$ et $AADL\_Project$. Il est possible de définir de nouveaux property sets.

- **Exemple :**

```
property set AADL_Properties is
   Deadline : aadlinteger
      applies to (thread, device, ...);
   Source_Text : inherit list of aadlstring
      applies to (data, port, thread, ...);
   ...
end AADL_Properties;
```

Outils de vérification (14)

- **Association de propriétés :**
  - Affecte une valeur à une propriété pour un composant donné.
  - Affectation dans l'implémentation et/ou le type d'un composant, par extension ou sur les instances.
- **Exemple :**

```
thread receiver
properties
  Compute_Execution_Time => 3 .. 4 ms;
  Period => 150 ms;
end receiver;

thread implementation receiver.impl
properties
   Deadline => 150 ms;
end receiver.impl;
```

Outils de vérification (15)

- **Connexions entre composants :** modélisent leurs interactions => flot de contrôle et/ou d'échange de données.
  - **Features :** interface du composant. Chaque feature est caractérisée par un nom, une catégorie, une orientation, un type de donnée, ...
  - **Catégories de feature** = types d'interaction :
    - event port : émission/réception d'un signal.
    - data port/event data port : échange synchrone/asynchrone d'un message.
    - subprogram parameter : paramètre lors d'appels de sous-programme.
    - data access : accès à une donnée partagée (composant data).
    - subprogram access : mise en oeuvre des RPCs.
    - ...

Outils de vérification (16)

- **Connexion pour accès à une donnée partagée :**

```
process implementation processing.others
   subcomponents
      analyse : thread analyser.impl;
      display : thread display_panel.impl;
      a_data  : data shared_var.impl;
   connections
      data a_data -> display.share;
      data a_data -> analyse.share;
end processing.others;

data shared_var;
end shared_var;
data  implementation shared_var.impl
end shared_var.impl;

thread analyser
features
 share : requires data access shared_var.impl;
end analyser;
```

Outils de vérification (17)

- **Que peut on attendre d'un modèle AADL :**
  1. Génération du code et de la documentation (ex : Ocarina de Télécom-Paris-Tech, STOOD d'Ellidiss Technologies).
  2. Analyse : sémantique, fiabilité, performances, ordonnançabilité, ...

- **Services offerts par Cheddar :**
  1. Analyse d'ordonnançabilité.
  2. Analyse d'empreinte mémoire.

Sommaire

Résumé

1. Algorithmes classiques pour le temps réel : priorité fixe, EDF. Techniques de vérification a priori d'un jeu de tâches.

2. Majorité des systèmes d'exploitation = philosophie à la POSIX 1003.1b : priorités fixes multi-files + FIFO et/ou round-robin.

3. Exemple de techniques pour la prise en compte des dépendances, du partage des ressources.

4. Liens entre architecture logicielle, matérielle et ordonnancement temps réel.

Sommaire

Références (1)

[BLA 76] J. Blazewicz. « Scheduling Dependant Tasks with Different Arrival Times to Meet Deadlines ». In. Gelende. H. Beilner (eds), Modeling and Performance Evaluation of Computer Systems, Amsterdam, Noth-Holland, 1976.

[CHE 90] H. Chetto, M. Silly, and T. Bouchentouf. « Dynamic Scheduling of Real-time Tasks Under Precedence Constraints ». *Real Time Systems, The International Journal of Time-Critical Computing Systems*, 2(3):181–194, September 1990.

[GAL 95] B. O. Gallmeister. *POSIX 4 : Programming for the Real World* . O'Reilly and Associates, January 1995.

[LIU 73] C. L. Liu and J. W. Layland. « Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environnment ». *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[SHA 90] L. Sha, R. Rajkumar, and J.P. Lehoczky. « Priority Inheritance Protocols : An Approach to real-time Synchronization ». *IEEE Transactions on computers*, 39(9):1175–1185, 1990.

Références (2)

[TIN 92] K. Tindell. « Using Offset Information to Analyse Static Priority Pre-Emptively Scheduled Task Sets ». In *YCS. 182, Dept of Computer Science, University of York*, 1992.

[TIN 94] K. W. Tindell and J. Clark. « Holistic schedulability analysis for distributed hard real-time systems ». *Microprocessing and Microprogramming*, 40(2-3):117–134, April 1994.

[XU 90] J. Xu and D. Parnas. « Scheduling Processes with Release Times, Deadlines, Precedence, and Exclusion Relations ». *IEEE Transactions on Software Engineering*, 16(3):360–369, March 1990.