

# Ordonnancement temps réel monoprocesseur

Département/IUP informatique

**Frank Singhoff**

**C-208**

**singhoff@univ-brest.fr**

## Sommaire

1. Introduction et concepts de base.
2. Algorithmes classiques pour le temps réel.
3. Un peu de pratique.
4. Prise en compte de dépendances.
5. Résumé.
6. Références.

# Partie 1

## Introduction et concepts de base

### Ordonnancement, définitions (1)

- **Objectifs** : prendre en compte les besoins d'urgence, d'importance et de réactivité des applications temps réel.
- **Éléments de taxinomie** :
  - Algorithmes hors ligne/en ligne : moment où sont effectués les choix d'allocation. Algorithmes statiques/dynamiques.
  - Priorités statiques/dynamiques : les priorités changent elles ?
  - 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)

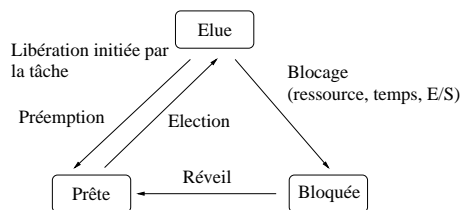
- Propriétés recherchées :

1. **Faisabilité** : est il possible d'exhiber un test de faisabilité ?
  - Condition permettant de décider hors ligne du respect des contraintes des tâches.
  - Prédicibilité du 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 ?, etc
4. **Facilité de mise en œuvre** : l'ordonnanceur est-il facile à implanter ?

## Méthode d'analyse

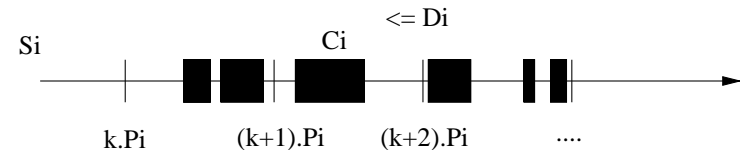
- Lorsqu'un problème d'ordonnancement se pose, on :
  1. On modélise les tâches du système et leurs contraintes temporelles.
  2. On choisit un algorithme d'ordonnancement.
  3. On valide la faisabilité du jeu de tâches : faisabilité théorique (ordonnançabilité et complexité) et faisabilité pratique (ordonnanceur facilement implantable, contraintes logiciels/matérielles, etc)  
Eventuellement, on retourne en 1 ou 2.

## Notion de tâche (1)



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

## Notion de tâche (2)



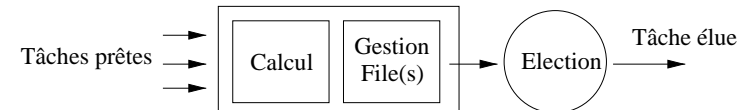
### • Paramètres définissant une tâche $i$ :

- Arrivée de la tâche dans le système :  $S_i$ .
- Borne sur le 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 :  $R_i$ .
- Tâche apériodique définie par  $(S_i, C_i, D_i, R_i)$  ; Tâche périodique  $(S_i, C_i, D_i, P_i)$  (idem pour sporadique mais  $P_i$  = un délai minimal inter-activations).

### Notion de t che (3)

- Un mod le simplifi  de t ches : **le mod le de Lui et Layland[LIU 73] ou mod le de t ches p riodiques    ch ance sur requ te.**
- Caract ristiques :
  - T ches p riodiques.
  - T ches ind pendantes.
  - avec  $\forall i : S_i = 0 \implies$  instant critique (pire cas).
  - avec  $\forall i : P_i = D_i \implies$  t ches    ch ances sur requ tes.

### Ordonnanceur : structure et fonctions (1)



#### • Trois traitements successifs :

- (1) Calcul d'informations d'ordonnancement :
  - En ligne ou hors ligne.
  - Calendrier : activation cyclique.
  - P riode : Rate Monotonic (RM).
  -  ch ance : Deadline Monotonic (DM), Earliest Deadline First (EDF).
  - Laxit  : Least Laxity First (LLF). (laxit    l'instant  $t$  :  $L_i(t) = D_i(t) -$  reliquat de  $C_i$    ex cuter).
  - Etc.

## Ordonnanceur : structure et fonctions (2)

### (2) Gestion de la/les files d'attente :

- Une file par priorité/importance/classe d'applications.
- Politique FIFO (déplacée si bloquée, libération à l'initiative de la tâche, terminaison).
- Politique round-robin (déplacement identique à FIFO + fin de quantum ; utilisation d'un ou plusieurs quantum).
- Politiques hybrides (ex : FIFO + round-robin)  
⇒ POSIX 1003.1b, Chorus, Solaris, etc ...

### (3) Phase d'élection :

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

## Partie 2

### Algorithmes classiques pour le temps réel

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

## L'algorithme Rate Monotonic (1)

### • Caractéristiques :

- Priorités fixes  $\Rightarrow$  analyse hors ligne  $\Rightarrow$  applications statiques.
- Tâches périodiques uniquement. Dans ce cours, pour RM, nous nous limitons à des tâches de type "Lui et Layland".
- Complexité faible et mise en œuvre facile dans un système d'exploitation.
- Algorithme optimal dans la classe des algorithmes à priorité fixe.

### • Fonctionnement :

1. Phase de calcul : priorité = inverse de la périodicité.
2. Phase d'élection : election de la plus forte priorité.

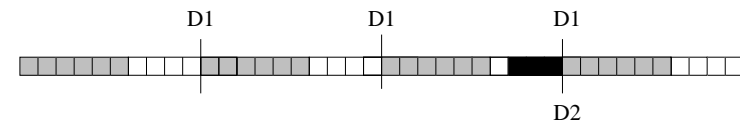
## L'algorithme Rate Monotonic (2)

### • Cas préemptif :

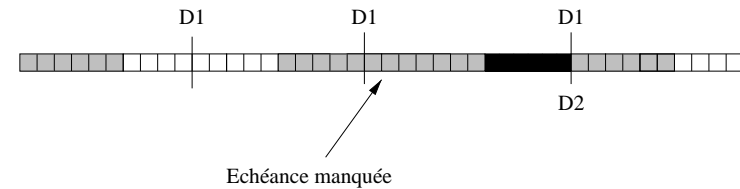
T1 : C1=6 ; P1=10 (gris)

T2 : C2=9 ; P2=30 (blanc)

Noir=libre



### • Cas non préemptif :



### L'algorithme Rate Monotonic (3)

- Evaluation de l'ordonnançabilité :

1. **Période d'étude** =  $[0, PPCM(P_i)]$ . Solution exacte.

2. **Taux d'utilisation** (cas préemptif) :

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

Condition suffisante mais non nécessaire : solution pessimiste donc.

3. **Temps de réponse**  $\implies$  délai entre l'activation d'un tâche et sa terminaison. Solution parfois exacte (selon les modèles de tâches).

### L'algorithme Rate Monotonic (4)

- **Calcul du 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 :

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

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

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

## L'algorithme Rate Monotonic (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  $w_i^{n+1} = w_i^n$ .

## L'algorithme Rate Monotonic (6)

- **Exemple** :

- Tâche 1 :  $P_1=7$ ;  $C_1=3$
- Tâche 2 :  $P_2=12$ ;  $C_2=2$
- Tâche 3 :  $P_3=20$ ;  $C_3=5$

- $w_1^0 = 3 \implies TR_1 = 3$
- $w_2^0 = 2$
- $w_2^1 = 2 + \left\lceil \frac{2}{7} \right\rceil 3 = 5$
- $w_2^2 = 2 + \left\lceil \frac{5}{7} \right\rceil 3 = 5 \implies TR_2 = 5$
- $w_3^0 = 5$
- $w_3^1 = 5 + \left\lceil \frac{5}{7} \right\rceil 3 + \left\lceil \frac{5}{12} \right\rceil 2 = 10$
- $w_3^2 = 5 + \left\lceil \frac{10}{7} \right\rceil 3 + \left\lceil \frac{10}{12} \right\rceil 2 = 13$
- $w_3^3 = 5 + \left\lceil \frac{13}{7} \right\rceil 3 + \left\lceil \frac{13}{12} \right\rceil 2 = 15$
- $w_3^4 = 5 + \left\lceil \frac{15}{7} \right\rceil 3 + \left\lceil \frac{15}{12} \right\rceil 2 = 18$
- $w_3^5 = 5 + \left\lceil \frac{18}{7} \right\rceil 3 + \left\lceil \frac{18}{12} \right\rceil 2 = 18$
- $\implies TR_3 = 18$

## L'algorithme Rate Monotonic (7)

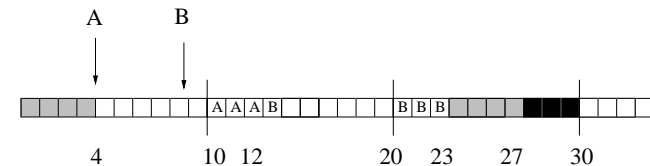
• Comment faire cohabiter des tâches apériodiques dans un système ordonnancé avec Rate Monotonic :

1. Les tâches apériodiques sont non critiques  $\Rightarrow$  priorités plus faible que les tâches périodiques.
2. Les tâches apériodiques sont critiques  $\Rightarrow$  utilisation de serveur de tâches apériodiques.

## L'algorithme Rate Monotonic (8)

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

T1 : C1=4 ; P1=20 (gris)      Noir=libre  
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



- 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 capacité gaspillée.
- Autres serveurs : serveur sporadique, différé, ...

## L'algorithme EDF (1)

### • Caract risques :

- Algorithme   priorit  dynamique  $\implies$  mieux adapt  que RM aux applications dynamiques.
- Supporte les t ches p riodiques et les t ches ap riodiques.
- Algorithme optimal : utilise jusqu'  100 pourcent de la ressource processeur.
- Mise en oeuvre difficile dans un syst me d'exploitation.
- Instable en sur-charge : moins d terministe que RM.

## L'algorithme EDF (2)

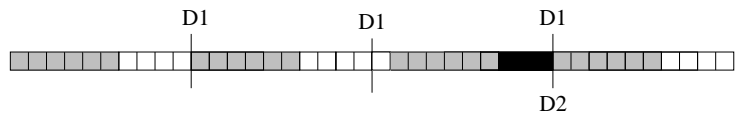
### • Fonctionnement :

1. Phase de calcul  $\implies$  calcul d'une  ch ance . Soit  $D_i(t)$ , l' ch ance   l'instant  $t$  et  $D_i$ , le d lai critique de la t che  $i$  :
  - T che ap riodique :  $D_i(t) = D_i + S_i$ .
  - T che p riodique :  $D_i(t) = \text{date de d but de l'activation courante   l'instant } t + D_i$ .
2. Phase d' lection :  lection de la plus courte  ch ance d'abord.

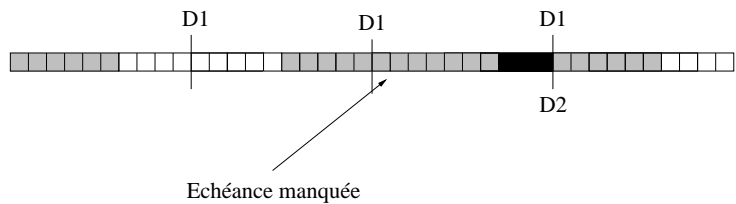
### L'algorithme EDF (3)

- Cas préemptif :

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



- Cas non préemptif :



### L'algorithme EDF (4)

- Evaluation de l'ordonnançabilité :

- Période d'étude** : idem Rate Monotonic **si tâches périodiques uniquement.**

- Taux d'utilisation**, cas préemptif, tâches périodiques et indépendantes avec  $\forall i : S_i = 0$  :

- 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  $\exists i : D_i < P_i$  :

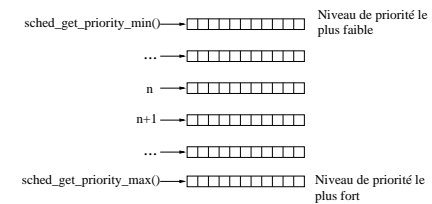
$$U = \sum_{i=1}^n \frac{C_i}{D_i} \leq 1$$

- Temps de réponse** : complexité importante.

## Partie 3

### Un peu de pratique

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

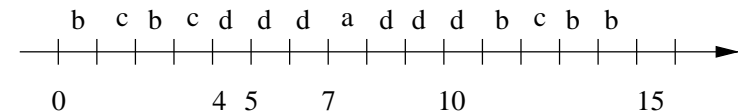
## La norme POSIX 1003.1b (2)

- Gestion des files d'attente POSIX : élection de la tâche en tête de file d'attente de plus haute priorité.
- Différences entre les politiques :
  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
<i>a</i>	1	7	1	FIFO
<i>b</i>	5	0	4	RR
<i>c</i>	3	0	4	RR
<i>d</i>	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 : h ritage par *fork()*, d marrage en section critique.
- Exemple : cf. t ches 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");
```

## Partie 4

### Prise en compte des d pendances : exemples de solutions

1. Contraintes de pr c dence (ex : communications).
2. Partage de ressources.

## Contraintes de précédence (1)

- Objectifs :

1. Calculer un ordonnancement qui respecte les contraintes de précédence (simulation, exécution).
2. Décider de la faisabilité hors ligne.

## Contraintes de précédence (2)

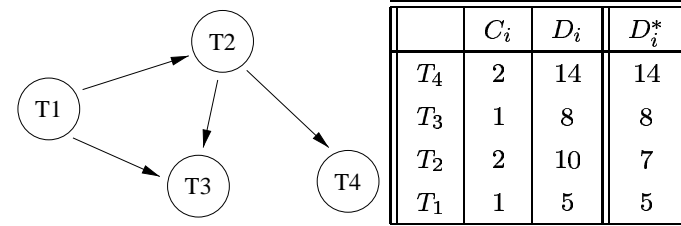
- Principales approches :

1. **Conditions initiales** (paramètre  $S_i$ ). Exécution et faisabilité (avec formules spécifiques).
2. **Affectation des priorités** (Chetto/Blazewicz [BLA 76, CHE 90]). Applicabilité limitée. Faisabilité et exécution.
3. **Modifications des délais critiques** (Chetto/Blazewicz). Applicabilité limitée. Faisabilité et exécution.
4. **Utilisation du paramètre "Jitter"** [TIN 94]. Seulement pour la faisabilité (pire cas éventuellement très grand).
5. **Heuristique d'ordonnancement** (Xu et Parnas [XU 90]). Pas de faisabilité.

## Contraintes de pr cedence (3)

- Principe de la solution de Blazewicz [BLA 76] et de Chetto et al [CHE 90] : rendre les t ches ind pendantes en modifiant leurs param tres.
- Hypoth ses : T ches soit ap riodiques, soit p riodiques de m me p riode.
- Technique :
  1. Modification pour RM :
    - $\forall i, j \mid i \prec j : \text{priorite}_i > \text{priorite}_j$
  2. Modification pour EDF :
    - $D_i^* = \min(D_i, \min(\forall j \mid i \prec j : D_j^* - C_j))$ .

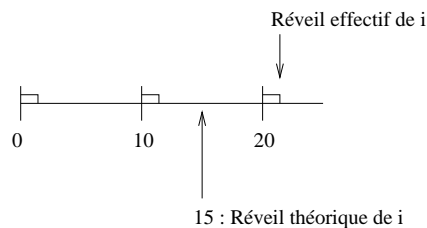
## Contraintes de pr cedence (4)



- Exemple : EDF + t ches ap riodiques.
  - $D_4^* = 14$ ;
  - $D_3^* = 8$ ;
  - $D_2^* = \min(D_2, D_3^* - C_3, D_4^* - C_4) = \min(10, 8 - 1, 14 - 2) = 7$ ;
  - $D_1^* = \min(D_1, D_2^* - C_2, D_3^* - C_3) = \min(5, 7 - 2, 8 - 1) = 5$ ;

## Contraintes de précédence (5)

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



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

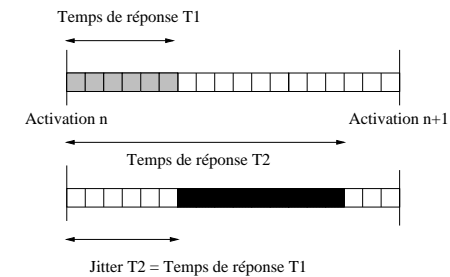
- Temps de réponse =  $r_i = w_i + J_i$ , avec :

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

## Contraintes de précédence (6)

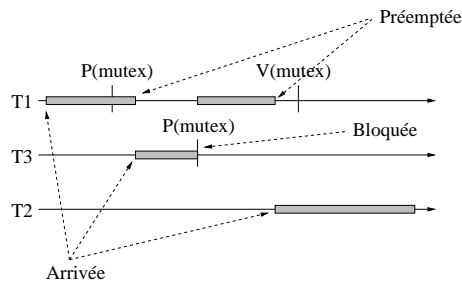
T1 :  $C1=6$  ;  $P1=18$  (gris)

T2 :  $C2=9$  ;  $P2=18$  (noir)



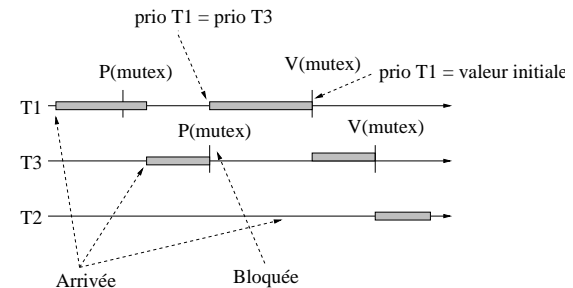
- Exemple du producteur/consommateur :
  - T1 et T2 sont activées toutes les 18 unités de temps.
  - T1 lit un capteur et transmet la valeur vers T2 qui doit l'afficher à l'écran.
  - T2 doit être activée sur terminaison de T1.
  - Quel est le temps de réponse de T2 ?

## Partage de ressources (1)



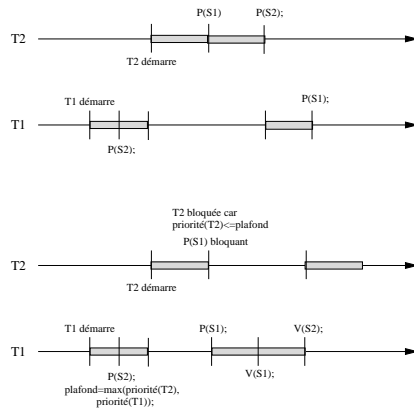
- Accéder à une ressource c'est éventuellement devoir attendre qu'elle se libère = borne sur le temps de blocage (noté  $B_i$ ).
- Problèmes :
  - Comment éviter les attentes non bornées?  $\Rightarrow$  protocoles d'évitement d'inversion de priorité.
  - Comment finement évaluer  $B_i$ ?
- Caractéristiques 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)



• Héritage plafond (ou Priority Ceiling Protocol ou PCP)[SHA 90] :

- Une variable (plafond) stocke le plus haut niveau de priorité de toutes les sections critiques actuellement acquises. P(sem) bloquant si priorité de la tâche  $\leq$  plafond.
- Ressources multiples sans interblocage.
- $B_i$  = plus grande section critique.

### Partage de ressources (4)

• Soit  $n$  tâches de type "Lui et Layland" 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 de type "Lui et Layland", ordonnancées par RM/préemptif :

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

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.**

## Partie 5

### Résumé

## Résumé

1. Algorithmes classiques pour le temps réel (calendrier, RM  $\Rightarrow$  système d'exploitation, EDF  $\Rightarrow$  applicatifs). Techniques de validation 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.
- Outil de simulation **Cheddar** :  
<http://beru.univ-brest.fr/~singhoff/cheddar>

## Partie 6

### Références

- [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, North-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 Environment ». *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.
- [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.