3. CONCEPTS ET MODÈLES POUR LE TEMPS RÉEL ASYNCHRONE ORDONNANCEMENT TEMPS RÉEL DES TÂCHES

3.1. MODÈLE CANONIQUE ET CONFIGURATION DE TÂCHES

Applications temps réel : Nature des tâches

Rythme d'occurrence

Tâche périodique Tp
Ex : lecture de capteurs

Tâche apériodique Tap
Ex : alarme
Tâche sporadique

Contraintes de temps

Tâche à contraintes strictes
Date de fin d'exécution au plus tard
Respect obligatoire **temps réel "dur"**

Tâche à contraintes relatives
Date de fin d'exécution au plus tard
Respect souhaitable **temps réel"mou"**

$T(r_0, C, R, P)$

avec $\leq C \leq R \leq P$

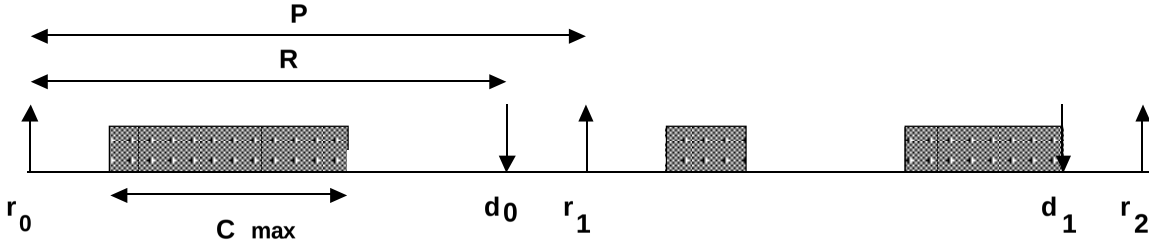
r_0 : date de réveil de la tâche
C : durée d 'exécution maximale
R : délai critique
P : période d 'exécution

r_k : date de réveil de la kième instance de la tâche $r_k = r_0 + kP$

d_k : échéance $d_k = r_k + R$

quand R = P : tâche à échéance sur requête

Diagramme temporel d 'exécution



Modèle canonique d'une tâche périodique temps réel

QUELQUES DÉFINITIONS UTILES

ORDONNANCEMENT D'UNE CONFIGURATION DE TÂCHES :
Détermination d'une séquence de planification des tâches

PROBLÈME DE L'ORDONNANCEMENT TEMPS RÉEL :
• En fonctionnement nominal, assurer, pour toutes les tâches, le respect des contraintes temporelles spécifiées
• En fonctionnement anormal (pannes, événements imprévus), atténuer les effets des surcharges temporelles et maintenir le procédé dans un état sécuritaire : respect des contraintes temporelles pour les tâches primordiales et meilleur effort pour les autres (mode dégradé).

TÂCHES INDÉPENDANTES :
Les tâches ne partagent que l'unité centrale.

TÂCHES DÉPENDANTES :
Les tâches partagent d'autres ressources ou sont reliées par des contraintes de précédence.

ORDONNANCEMENT PRÉEMPTIF OU NON
Dans le cas préemptif une tâche peut perdre le processeur au profit d'une tâche plus urgente.
Un ordonnanceur non préemptif n'arrête pas l'exécution d'une tâche élue.

ORDONNANCEMENT HORS LIGNE :
Séquence de planification construite avant le démarrage à partir des paramètres de toutes les tâches.
=> À l'exécution, une seule tâche cyclique ou séquence de tâches fixée par une table d'élection

ORDONNANCEMENT EN LIGNE
Séquence construite dynamiquement pour les tâches déclenchées et exécution avec ou sans préemption

QUELQUES DÉFINITIONS UTILES (suite)

LAXITÉ DU PROCESSEUR À UNE DATE t (ou encore MARGE À UNE DATE t) :
Durée maximale pendant laquelle le processeur peut rester inactif, à t, sans perdre l'ordonnançabilité.
Pour respecter les contraintes strictes, la laxité ne doit jamais devenir négative.
La laxité dépend de la séquence d'ordonnancement produite par l'algorithme d'ordonnancement utilisé.

SÉQUENCE VALIDE :

Séquence d'ordonnancement telle que toutes les tâches de la configuration respectent leurs échéances

ALGORITHME OPTIMAL :

Algoritme d'ordonnancement capable de produire une séquence valide pour toute configuration ordonnançable

TEST D'ORDONNANÇABILITÉ
Sert à vérifier qu'une configuration de tâches périodiques peut respecter ses contraintes temporelles.

TEST D'ACCEPTABILITÉ (ou test de garantie) D'UNE NOUVELLE TÂCHE
Vérifie qu'en ajoutant une nouvelle tâche à une séquence dynamique, celle-ci reste ordonnançable

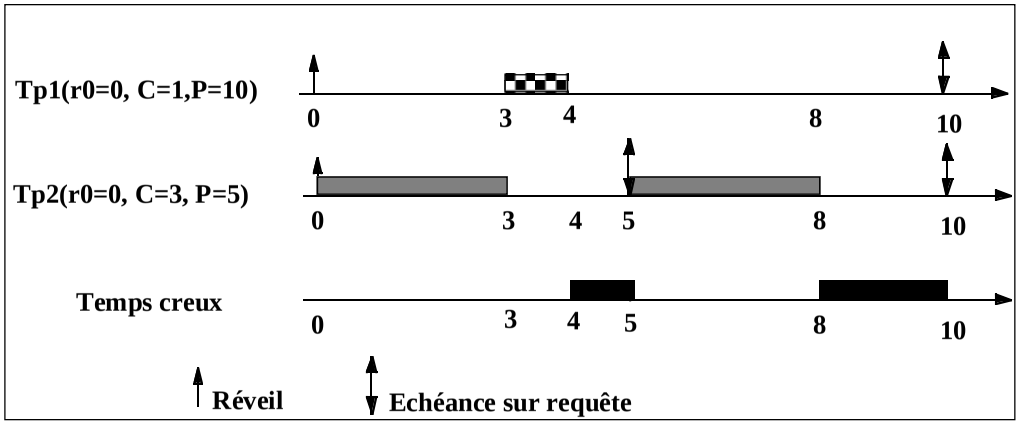
AUTRES INDICATEURS

TEMPS DE RÉPONSE : Délai écoulé entre le déclenchement et la fin d'exécution d'une tâche
utile pour une tâche apériodique à contraintes relatives

GIGUE : Dispersion autour de la période moyenne. (gigue de démarrage, de fin, du temps de réponse)
utile lorsque la régularité de certaines actions périodiques est demandée

3.2. ORDONNANCEMENT DE TÂCHES INDÉPENDANTES

3.2.1. ALGORITHME À PRIORITÉS FIXES



ORDONNANCEMENT "RATE MONOTONIC" DE TÂCHES TEMPS RÉEL PÉRIODIQUES
La tâche de plus petite période est la plus prioritaire

TÂCHES À ÉCHÉANCE SUR REQUÊTE : R = P

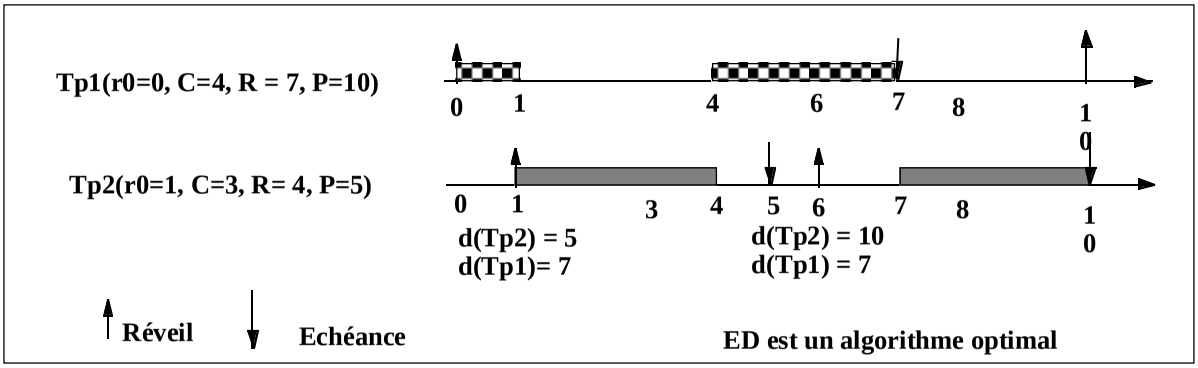
Condition suffisante d'ordonnançabilité d'une configuration de tâches à échéance sur requête

$$\sum Ci/Pi \leq n(2^{1/n} - 1)$$

autres algorithmes à priorités fixes : priorités empiriques, "deadline monotonic" (selon délai critique)

ORDONNANCEMENT DE TÂCHES INDÉPENDANTES

3.1.2. ALGORITHME À PRIORITÉS VARIABLES



ORDONNANCEMENT "EARLIEST DEADLINE" DE TÂCHES TEMPS RÉEL PÉRIODIQUES

La tâche la plus proche de son échéance est la plus prioritaire

TÂCHES À ÉCHÉANCE SUR REQUÊTE :
Condition nécessaire et suffisante d'ordonnançabilité d'une configuration de tâches : $\sum Ci/Pi \leq 1$

TÂCHES À ÉCHÉANCES QUELCONQUES
Condition suffisante d'ordonnançabilité d'une configuration de tâches quelconques : $\sum Ci/Ri \leq 1$
Condition nécessaire d'ordonnançabilité d'une configuration de tâches quelconques : $\sum Ci/Pi \leq 1$

Noter que l'exemple n'est pas une configuration ordonnançable par "rate monotonic"

Autre algorithme à priorités variables : "least laxity first" (ou "least slack time"),

ORDONNANCEMENT DE TÂCHES INDÉPENDANTES

3.1.3. PRISE EN COMPTE DES TÂCHES APÉRIODIQUES

SERVEUR DE TÂCHES APÉRIODIQUES

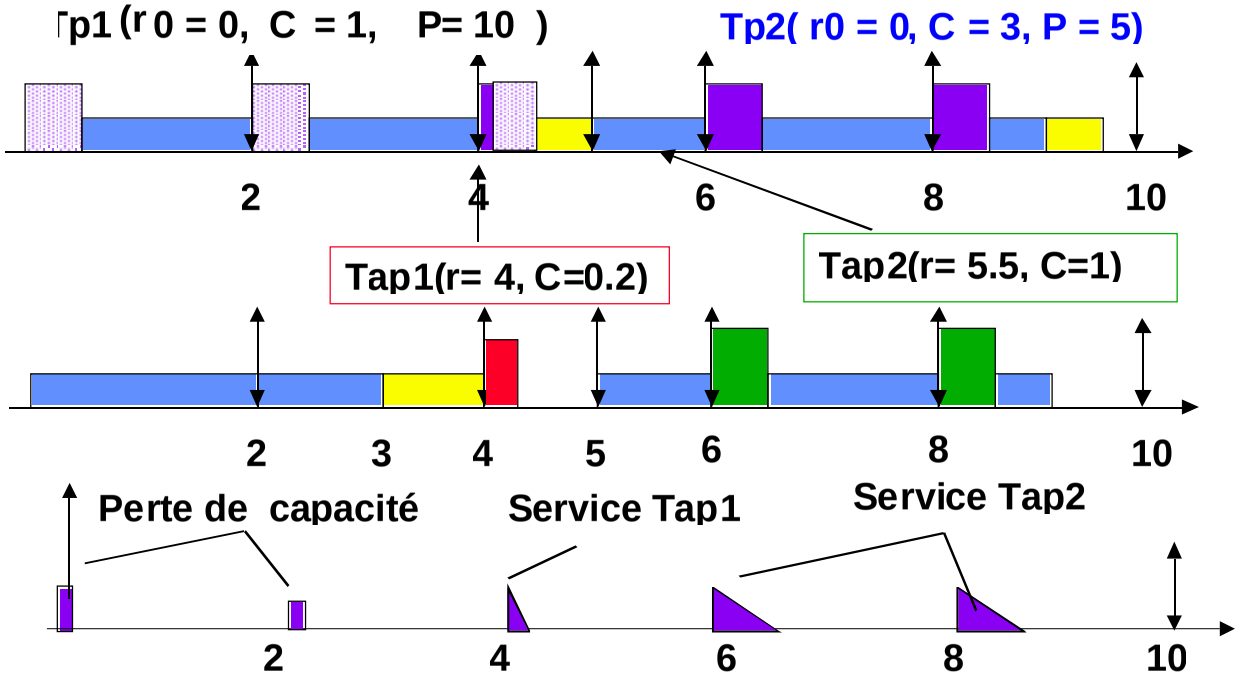
Un serveur périodique réserve du temps du processeur avec une période P et une capacité C et il est mis en place pour servir les tâches apériodiques déclenchées.
Ce serveur fait partie de la configuration périodique à ordonnancer.
Le serveur de scrutation, le serveur ajournable, le serveur à échange de priorité, le serveur sporadique.
Convient bien pour les tâches apériodiques à contraintes relatives quand on donne priorité aux tâches périodiques sur les apériodiques (on ne rejette pas, on retarde éventuellement en servant à l'ancienneté).
Le serveur sporadique est le serveur qui donne les meilleurs temps de réponse pour les tâches apériodiques.

TESTS D'ACCEPTABILITÉ

Au déclenchement d'une tâche périodique, on déroule une routine de garantie pour vérifier si la nouvelle tâche peut s'exécuter en respectant ses propres contraintes temporelles ainsi que celles des tâches périodiques et celles des apériodiques précédemment acceptées et non encore terminées.
Tests des temps creux (test sur un préordonnancement des périodiques) ou test de laxité (test dynamique)
Convient bien quand on donne priorité aux tâches périodiques sur les apériodiques et pour les tâches apériodiques à contraintes strictes (peut refuser des tâches apériodiques)

Problème : et si on veut que certaines tâches apériodiques soient plus prioritaires que certaines tâches périodiques ? ==> introduire un paramètre d'importance pour gérer les surcharges

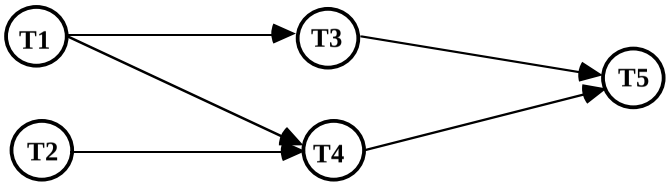
Ordonnancement des tâches apériodiques
Serveur de scrutation : Tpsc(r_0=0, C=0.5, P=2)



3.3. ORDONNANCEMENT DE TÂCHES DÉPENDANTES

3.3.1. TÂCHES AVEC CONTRAINTES DE PRÉCÉDENCE

• Graphe de précédence entre tâches d'une application et prise en compte des contraintes de précédence.



• Modifications des paramètres temporels et affectation des priorités (on a des formules par récurrence).

	Paramètres des tâches			*"Rate Monotonic"*		*"Earliest Deadline"*	
Tâche	r_i	C_i	d_i	r^*_i	$Prio_i$	r^*_i	d^*_i
T_1	0	1	5	0	2	0	3
T_2	5	2	7	5	1	5	7
T_3	0	2	5	0	3	1	5
T_4	0	1	10	5	3	7	9
T_5	0	3	12	5	4	8	12

Exemple, modifications pour la politique ED : une tâche n'est activable que si tous ses prédécesseurs ont terminé leur exécution. La date de réveil d'une tâche doit être supérieure à toutes les dates de réveil de ses prédecesseurs immédiats augmentées de leur durée d'exécution. L'échéance d'une tâche doit être inférieure à toutes les échéances de ses successeurs immédiats diminuées de leur temps d'exécution.

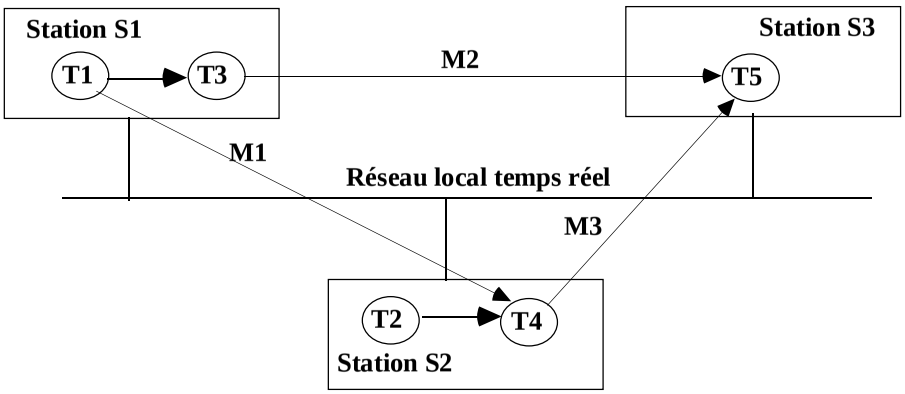
ORDONNANCEMENT DE TÂCHES DÉPENDANTES

3.3.2. ORDONNANCEMENT CONJOINT DE TÂCHES ET DE MESSAGES

• Cas de réseaux locaux temps réel dont on sait calculer les durées de transmission des messages avec des hypothèses réalistes (réseau fiable, horloges des divers sites suffisamment synchronisées).

On suppose qu'une tâche ne peut commencer son exécution que si tous les messages qu'elle attend sont reçus et qu'elle ne transmet ses messages qu'à la fin de son exécution.
Le transfert de message peut se traiter comme une relation de précédence entre tâches.

Exemple de placement de tâches d'une application temps réel.



3.3.2. ORDONNANCEMENT CONJOINT DE TÂCHES ET DE MESSAGES

Calculs pour l'exemple

On ne réveille pas une tâche si ses prédécesseurs n'ont pas eu le temps de faire leur durée d'exécution et de faire parvenir leur message. C'est ainsi qu'on prend en compte les messages. Les échéances sont définies avant l'envoi des messages. La durée des messages n'intervient pas dans la modification des échéances.

	Paramètres initiaux de tâches			Paramètres modifiés pour ED	
Tâche	ri	Ci	di	r*i	d*i
T1	0	1000	5000	0	3000
T2	5000	2000	7000	5000	7000
T3	0	2000	5000	1000	5000
T4	0	1000	10000	Max{1000 + D1,7000}	9000
T5	0	3000	12000	Max{3000 + D3, Max{2000 + D1+ D2, 8000}	12000

Exemple de calcul des paramètres pour l'application placée dans un réseau local temps réel

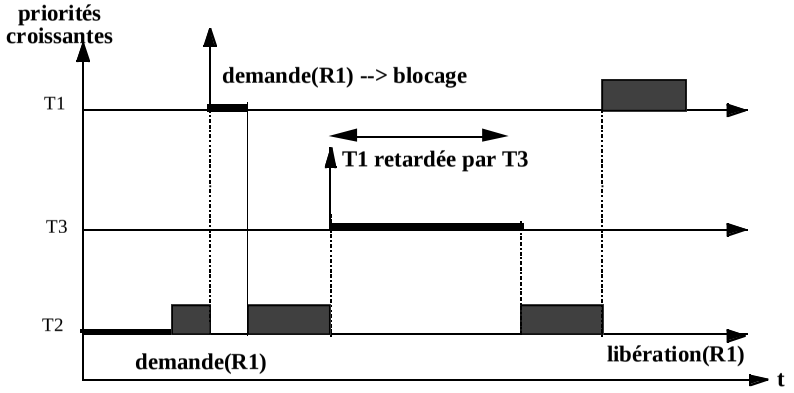
Message	Taille (en octets)	Durée de transmission (en microsecondes)			
			bus_à_jeton	FIP	CAN
M1	2	D1	112	178	73
M2	8	D2	160	226	130
M3	4	D3	128	194	92

Résultats des calculs de durées de transmission des messages

ORDONNANCEMENT DE TÂCHES DÉPENDANTES

3.3.3. TÂCHES PARTAGEANT DES RESSOURCES CRITIQUES

Le phénomène de l'inversion de priorité

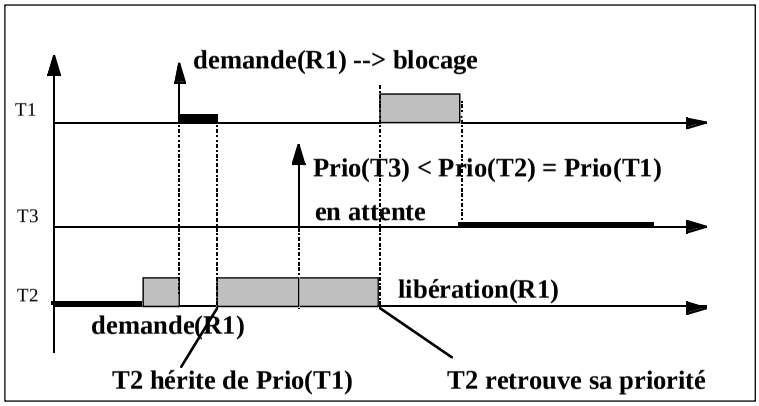


Inversion de priorité due au conflit entre priorité et accès exclusif à une ressource R1
T1 et T2, respectivement le processus de plus haute priorité et de plus faible priorité partagent R1
T3 se termine avant T1, bien que que T1 soit plus prioritaire

ORDONNANCEMENT DE TÂCHES DÉPENDANTES

TÂCHES PARTAGEANT DES RESSOURCES CRITIQUES

Prévention de l'inversion de priorité par héritage de priorité



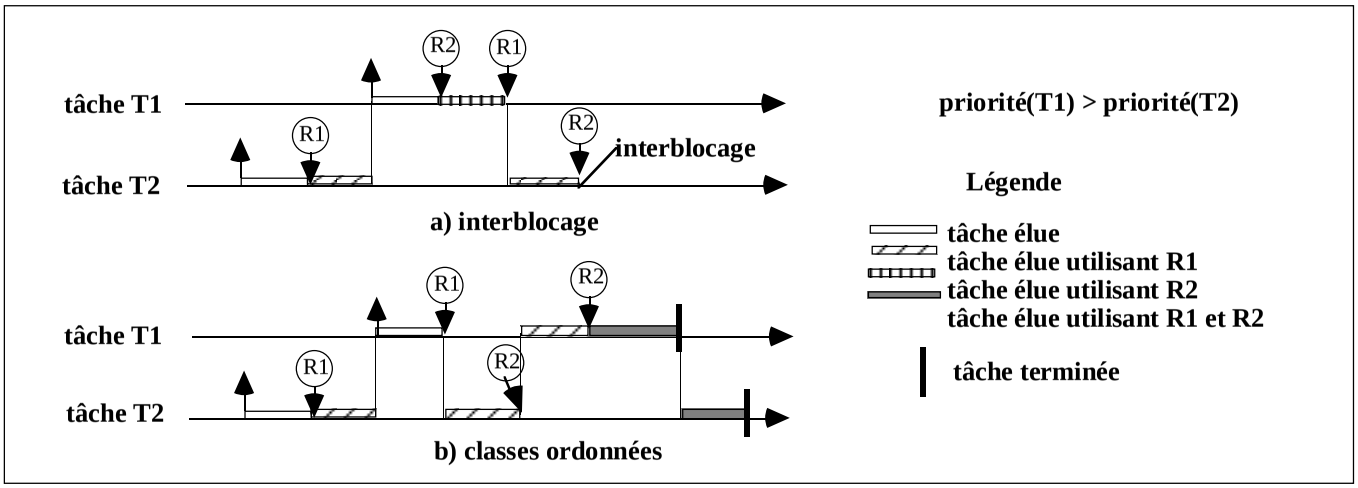
Prévention de l'inversion de priorité : Application du protocole de l'héritage de priorité

• protocole de l'héritage de priorité : tout processus en section critique augmente sa priorité en héritant de la priorité maximale des processus en attente de cette section critique (bien entendu si sa priorité est supérieure, il ne fait rien).
En sortant de section critique, chaque processus récupère la priorité qu'il avait auparavant.
• autres protocoles : verrou le plus haut, priorité plafonnée

ORDONNANCEMENT DE TÂCHES DÉPENDANTES

TÂCHES PARTAGEANT DES RESSOURCES CRITIQUES

problème de l'interblocage, solution par classes ordonnées

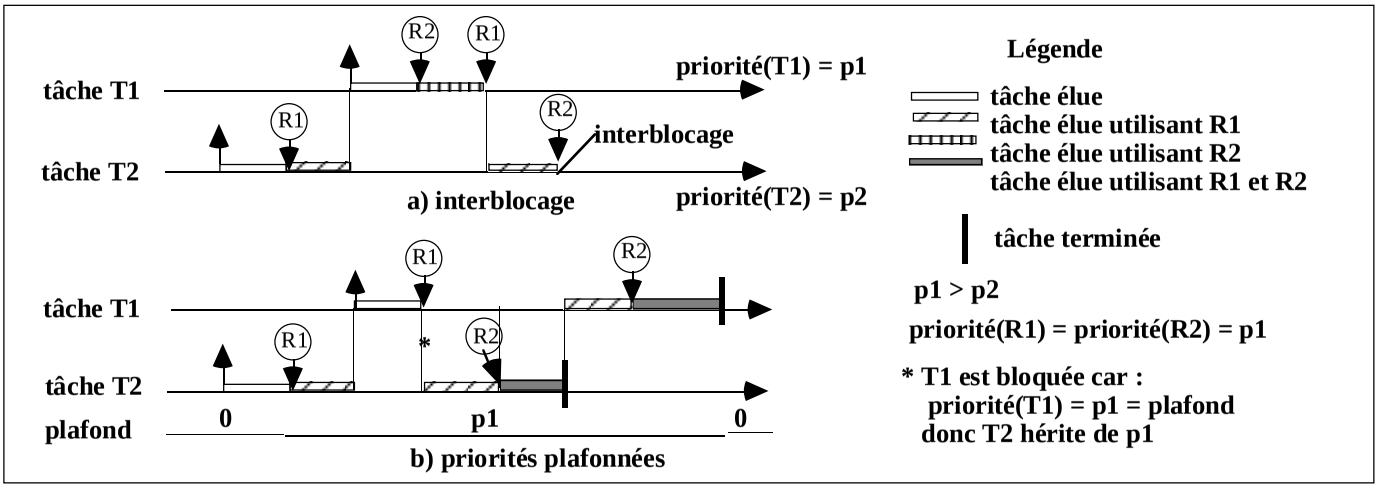


Exemple d'interblocage et de solution par la prise ordonnée des ressources
solution valable en monoprocesseur et en multiprocesseur

ORDONNANCEMENT DE TÂCHES DÉPENDANTES

TÂCHES PARTAGEANT DES RESSOURCES CRITIQUES

solution de l'interblocage et de l'inversion de priorité : priorité plafonnée



Chaque ressource possède une priorité, celle de la tâche de plus haute priorité qui peut la demander
Une tâche ne peut obtenir une ressource que si la priorité de la ressource est strictement supérieure au plafond courant = priorité maximale de toutes les ressources déjà allouées à d'autres tâches.
Une tâche qui possède la ressource hérite de la priorité de la tâche la plus prioritaire en attente.
Solution valable en monoprocesseur seulement

3.4. DIVERS ASPECTS AVANCÉS

3.4.1. ORDONNANCEMENT EN SITUATION DE SURCHARGES

Situation de surcharge : augmentation de la durée nominale C ou avalanche de tâches apériodiques.
Conséquences des surcharges : fautes temporelles

RM, EDF ne sont pas adaptés : propagation des dépassements, instabilité de l'ordonnancement (cas de EDF)

Politiques face aux surcharges :

•• Surcharge tolérable parfois si contraintes relatives.
Soit on ne fait rien, soit on utilise un coût de dépassement pour servir en minimisant ce coût ("best effort").
•• Surcharge intolérable, on doit récupérer du temps de processeur :
1. Arrêter les tâches qui ont dépassé leur échéance
2. Supprimer les tâches qui sont déclenchées après leur échéance
- **ce ne sont pas forcément les tâches fautives, c'est peut-être déjà trop tard (avalanche partie)**
- **ce sont peut-être des tâches primordiales pour l'application**

Avec un serveur périodique des tâches apériodiques, on supprime des tâches apériodiques (primordiales?)

3. Algorithmes à importance : l'urgence ne suffit pas, il faut un autre paramètre : l'importance
urgence ≠ importance
- **Importance = Ordre (ou un rang de suppression) ou Importance = pondération du coût de dépassement**
- **Notion émergente "feedback scheduling" : l'importance sert à réguler le nombre de tâches ordonnancées**

4. Dans un réseau, placement dynamiques de tâches les moins fréquentes (ou mieux, les moins importantes) sur les sites qui ont du temps de calcul disponible avant l'échéance des tâches à placer.
- **critère de placement dynamique : la charge ou la laxité (difficiles à évaluer avec précision)**

3.4. DIVERS ASPECTS AVANCÉS (suite)

3.4.2. ORDONNANCEMENT MULTIPROCESSEUR

problèmes très difficiles (NP complets) + quelques anomalies : attention !
L'accroissement du nombre des processeurs, la réduction de la durée d'exécution d'une tâche, la diminution des contraintes de précédence peuvent conduire à une augmentation de la longueur de la séquence, et partant, au dépassement de certaines échéances (vrai surtout si placement statique des tâches).
résultats théoriques : pas de d'optimalité en ligne
quelques heuristiques habituels : Earliest Deadline, Least Laxity
placement statique assez utilisé

3.4.3. PROBLÈMES AJOUTÉS PAR LA RÉPARTITION

- **Variabilité des délais de communication entre deux sites et d'un site à l'autre :**
  (variation dans le temps et dans l'espace)
- **Absence d'horloge commune et dérives entre les horloges locales : coût de la synchronisation d'horloge**
- **Pas de mémoire commune contenant un état de référence et une vue unique de l'application**
- **Fiabilité faible et coût temporel de l'obtention d'un consensus (si pas impossibilité)**

DIFFICULTÉS SUPPLÉMENTAIRES POUR LE TEMPS RÉEL

- **Mesures de dates ou délais trop imprécises,**
- **Retards dans l'observation de l'évolution du procédé**
  ==> perte de la stabilité de la commande (oscillations, divergence)
  ==> grand intervalle d'erreur pour les tests d'ordonnançabilité et les calculs de garantie

- **Tolérance aux fautes plus problématique car on doit intégrer le délai d'obtention de consensus réparti.**

3.4. DIVERS ASPECTS AVANCÉS (fin)

3.4.4. PLACEMENT STATIQUE HORS LIGNE

Placement de N tâches sur P processeurs avec des contraintes.
Contraintes de ressources, de coûts de communication de messages entre sites, d'échéances, ...
Problème NP complet. Heuristiques diverses : sac à dos, gradient, ...
Techniques de recherche opérationnelle pour combinatoire, réseaux de neurones,
Le placement est figé à la configuration du système

3.4.5. PLACEMENT DYNAMIQUE

Placement sans migration ou placement avec migration des tâches
Pour limiter le coût de migration, on place des requêtes de tâches mais ensuite la requête n'est plus déplacée
Le code est dupliqué pour ne pas à avoir à le transférer. Seul le contexte d'une tâche migre à chaque requête.
Méthode utilisée pour accroître la tolérance aux fautes du matériel et aussi aux fautes temporelles

3.4.5. TOLÉRANCE AUX FAUTES ET CONSENSUS

Vue d'ensemble de ce domaine émergent dans [Chevochot1999]
Notion de systèmes temps réel prévisibles : garantie de respect des spécifications temporelles [Lelann1992]
Quelques environnements expérimentaux de systèmes répartis temps réel stricts tolérant les fautes
MARS avec bus doublé et calculateurs triplés sur chaque noeud [Kopetz 1989], GUARDS [Wellings 1998]

3.4.6. VALIDATION

1) le temps réel = concurrence => validation des tâches concurrentes (non interblocage, non famine)
2) le temps réel = contraintes de temps => validation des temps de réponse et des échéances