Systèmes Temps Réel
Université de Lille - Master 2 GI-II

John Klein

Objectifs et finalités du module

- Comprendre les enjeux du temps réel,
- Maîtriser les implications du temps réel sur les systèmes,
- Découvrir quelques méthodes et solutions existantes,
- Mettre en oeuvre ces mécanismes en TPs.

Définitions

- James Martin :
  "Un calculateur temps réel peut être défini comme contrôlant son environnement en recevant les informations, les traitant et agissant en retournant les résultats suffisamment vite pour affecter, modifier l'environnement à chaque instant."

- Abrial-Bourgne :
  "Un système fonctionne en temps réel s'il est capable d'absorber toutes les informations d'entrée sans qu'elles soient trop vieilles pour l'intérêt qu'elles présentent, et par ailleurs, de réagir à celles-ci suffisamment vite pour que cette réaction ait un sens."

Définitions

Classification des systèmes informatiques :

- Systèmes clos : compilateur

- Systèmes interactifs :
  - Systèmes réactifs aux événements : formulaire web en PHP
  - Systèmes temps réel : ordinateur de bord

Définitions

Limites des systèmes classiques pour le temps réel :

- sourd au monde extérieur (pas conçus pour ça),
- incapable de réagir aux événements,
- pas d'outil permettant le respect de contraintes de temps.

Définitions

En somme :

- Contraintes : temporelles sur le système en plus des contraintes de résultat classiques
  - exemple de contrainte de temps : la température du réacteur d'une centrale nucléaire doit être actualisée toutes les X ms.
  - exemple de contrainte classique : les portes du métro doivent s'ouvrir après l'arrêt en station.
- Interaction avec l'environnement
- Périssabilité des résultats (date associée au résultat, notion de timeout)
- Attention ne pas confondre avec la vitesse d'exécution !
  - exemple : prévision météo, l'algorithme a 24h pour présenter une carte prévisionnelle $\rightarrow$ c'est du temps réel !

Définitions

Différents types de temps réel :

- Temps réel critique ou Hard real time : temps de réponse du système garanti sous peine d'effondrement du système,
- Temps réel non critique ou Soft real time : temps de réponse moyen du système garanti (contrainte statistique).

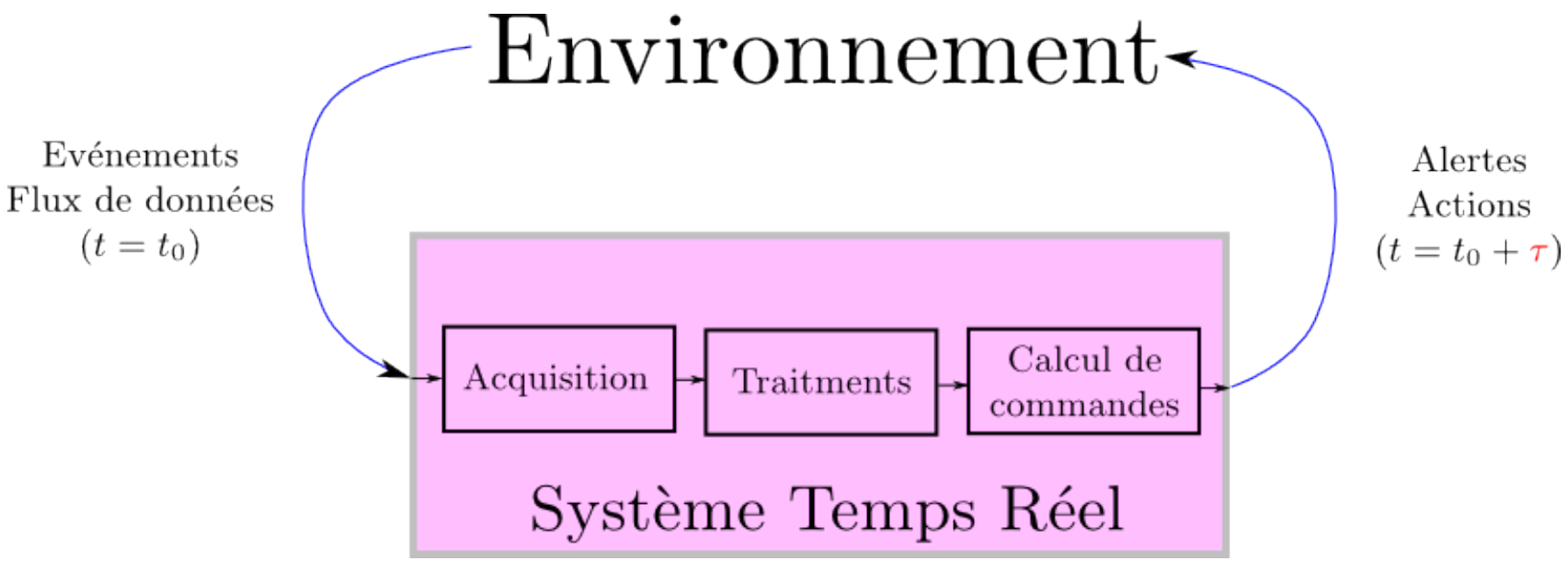
Définitions

- Remarque :
  Dans certains domaines applicatifs, on utilise une définition abusive du temps réel, par ex : traitements vidéo, temps réel = au moins 15 fps (on sous-entend que le traitement doit respecter une contrainte de temps suffisante pour ne pas gêner l'utilisateur).

- Vocabulaire connexe :
  système embarqué (embedded system) : il est intégré à un dispositif physique dont il gère le processus et dont il partage les contraintes environnementales.
  système partagé (distibuted system) : la couche matérielle et/ou logicielle est répartie en des lieux différents

Définitions

Un système temps réel, de par nature, communique avec son environnement :



environnement : opérateur humain, station relais, système maître, ...

Définitions

Quel est le travail d'un ingénieur temps réel ? Dimensionner le système

- analyser le cahier des charges et en dégager les contraintes
- structurer le système pour atteindre les contraintes de résultats
- évaluer la vitesse d'exécution nécessaire à chaque composant de la structure indépendamment de toute modification de l'environnement
- mettre en place des solutions de fonctionnement en cas d'erreur pour éviter l'arrêt complet du système ("le système doit retomber sur ses pattes")

En conséquence, le comportement du système doit être prévisible.

Spécification

Définition :
détermination des caractéristiques exigibles d'un système en fonction de l'analyse des besoins. Les exigences sont décrites en terme de :

- fonctionnalités
- contraintes
- performances

$\rightarrow$ c.f. cours UML pour un langage adapté à la problématique de la spécification.

Définitions

Contrainte de temps sur le système :

Temps de réponse t_r

- Temps entre l'apparition d'un événement à l'entrée du système et le fin du traitement de l'événement.
- Il faut prendre en compte
  - la valeur ou l'ordre de grandeur de t_r,
  - la possibilité ou non de faire évoluer t_r,
  - les répercussions sur le système en cas de non respect de la contrainte associée à t_r.

Définitions

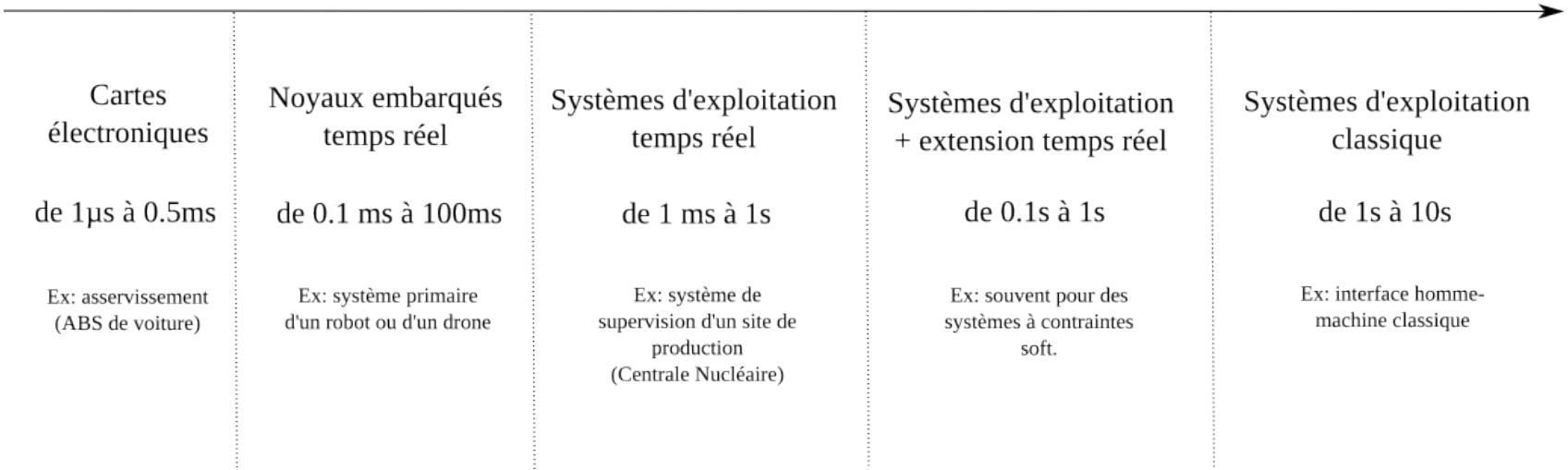
Il faut également étudier, en fonctions des différents états possibles du système, :

- le t_r moyen (soft),
- le t_r dans le pire cas (hard).

La contrainte de temps est alors : $t_r < t_{lim}$ où t_{lim} est le temps limite garantissant le bon fonctionnement de l'application temps réel.

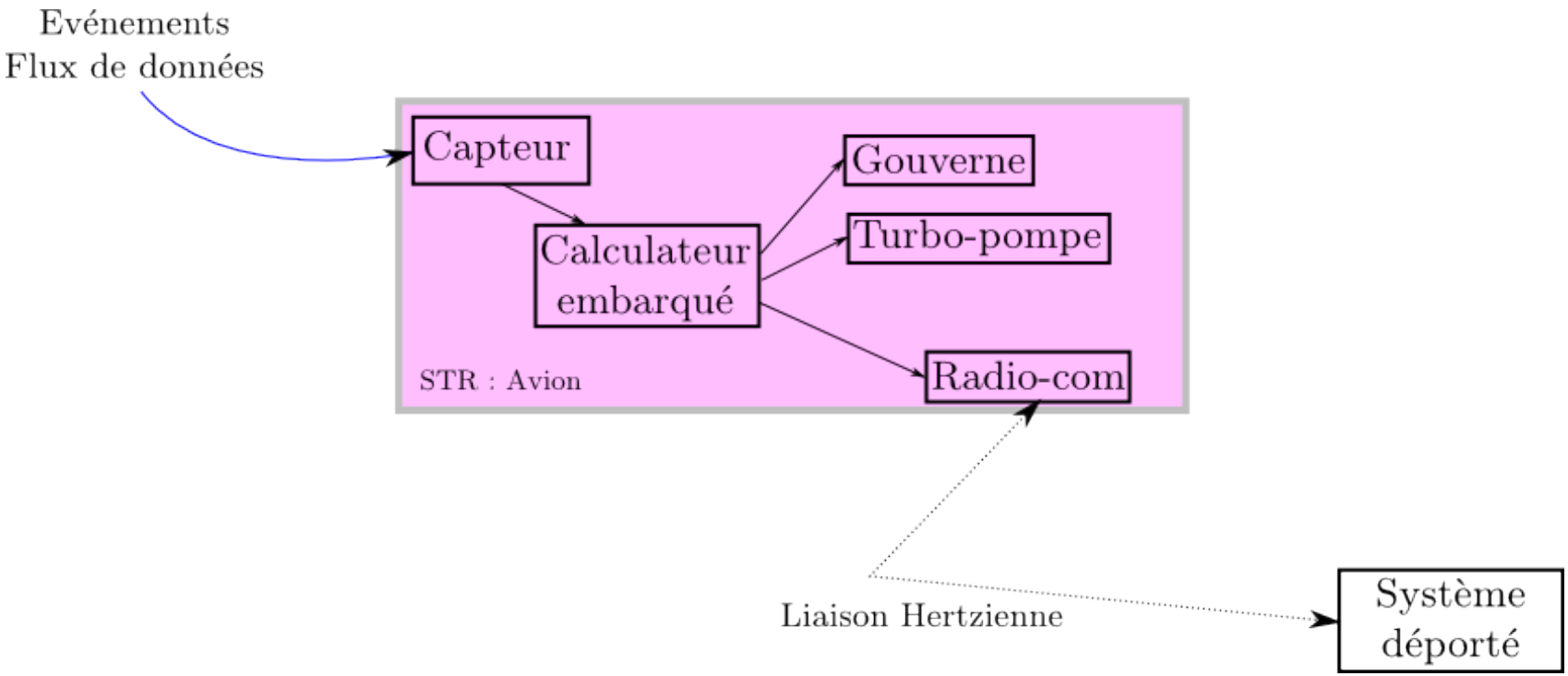
Définitions

Ordre de grandeur des temps de réponse de différents systèmes



Exemple

Pilotage temps réel d'un avion



Exemple

Imaginons 5 Fonctions à remplir pour ce STR :

- F1 : traitement des informations en provenance du sol (consignes de vol),
- F2 : envoi d'informations vers le sol (position, paramètres de fonctionnement),
- F3 : asservissement de la trajectoire (action sur les gouvernes),
- F4 : asservissement de la vitesse de rotation des turbo-pompes,
- F5 : prise en compte d'incidents.

Exemple

Les fonctions sont appelées sur événement ou de façon cyclique.

Fonction	Déclenchement	Contrainte de temps
F1	si données entrantes	$<$1s
F2	cyclique : toutes les 10s	$<$10s
F3	cyclique : toutes les 1s	$<$1s
F4	cyclique : toutes les 10ms	$<$10ms
F5	si incident	$<$1ms

Lors de la définition des contraintes, il est impératif de prendre en compte la durée de validité des informations !
(ex : si le calcul de la position prend un quart d'heure, l'information est périmée, l'avion a fait 200km entre deux).

Exemple

Solution : **un calculateur par fonction**

Avantages	Inconvénients
rapidité des traitements	coût
respect des contraintes de temps	encombrement et masse
	sous-utilisation des calculateurs (fonction F5 très rarement appelée)
	traitements supplémentaires nécessaires pour partager les résultats de chaque calculateur
	synchronisation des traitements plus difficile

Exemple

Solution : **un seul calculateur**

Avantages	Inconvénients
coût réduit	besoin d'une politique de gestion de l'UC
meilleure utilisation des ressources	
partage d'informations plus immédiat	
synchronisation des traitements plus facile	

Objectif

Comment optimiser l'utilisation de l'UC ?

- acheter un μp plus performant : meilleure fréquence horloge, architectures multi-coeur / multi-processeurs
  $\rightarrow$ solution coûteuse

- concevoir le système de sorte que l'UC soit le moins souvent inactive
  $\rightarrow$ solution gratuite

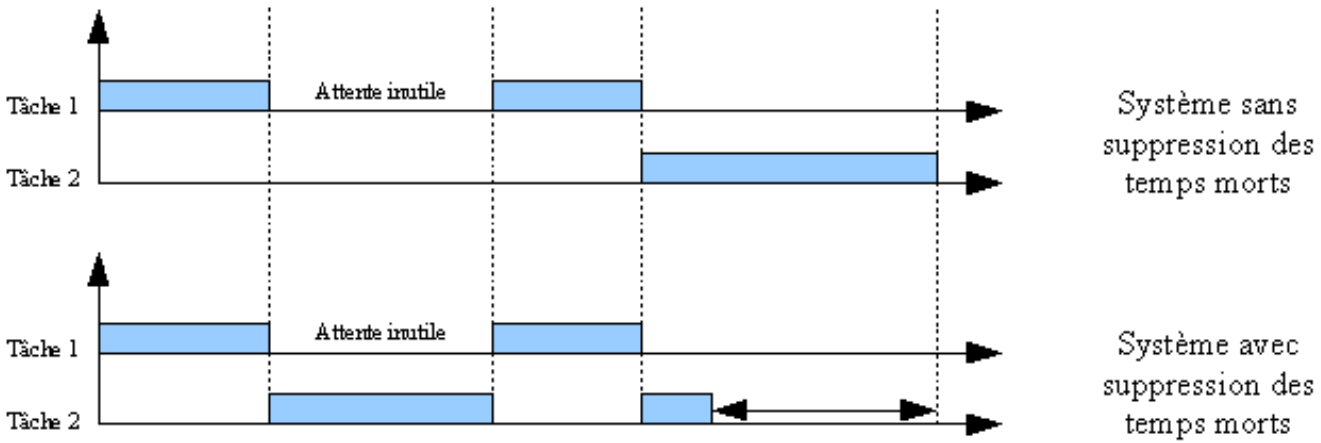
Objectif

Solution : le multi-tâches

- découper le processus en tâches,
- allouer l'UC d'une tâche à l'autre, si la première est "en attente".

Objectif

Exemple de programmation multi-tâches :



Objectif

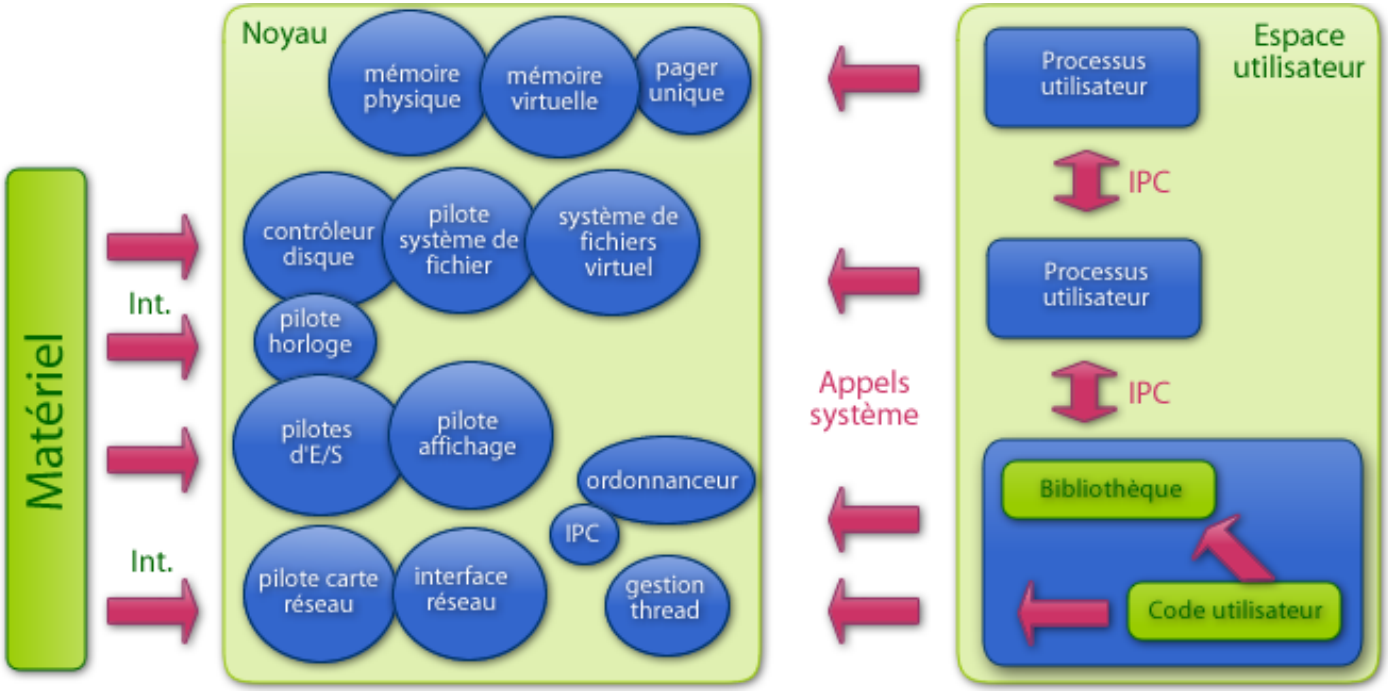
Suite du cours :

- Eléments d'architecture des ordinateurs : quelques rappels sur l'architecture d'un système informatisé en lien avec les exigences du multi-tâches et a fortiori du temps réel.
- Solutions informatiques pour mettre en oeuvre le multi-tâches et ce qui s'y rapporte (interruption événementielle, ordonnancement, communication entre tâches).

Système d'exploitation

Définition : (wikipedia)

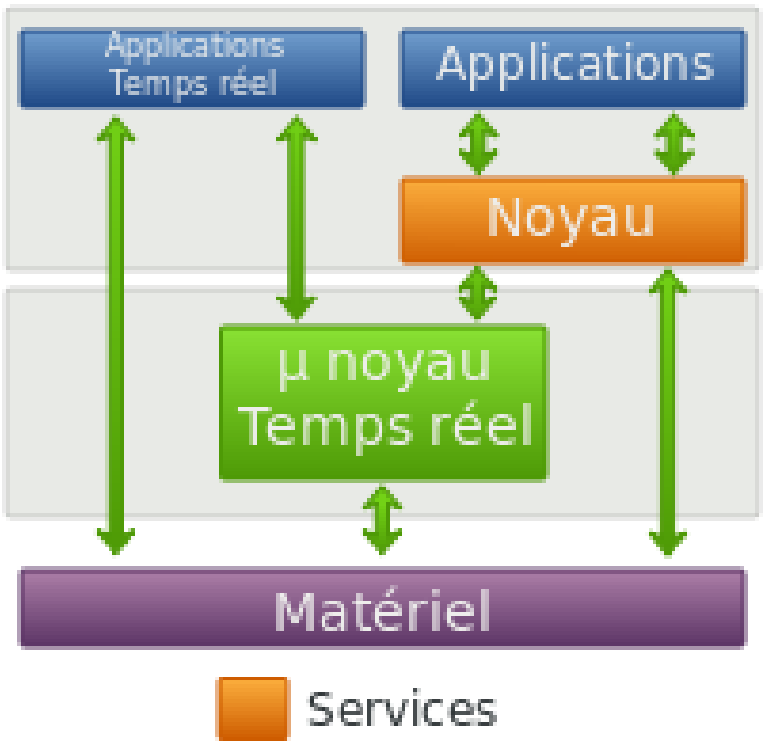
ensemble de programmes responsables de la liaison entre les ressources matérielles d'un ordinateur et les applications informatiques de l'utilisateur (traitement de texte, jeux vidéo, etc). Il fournit aux programmes d'applications des points d'entrée génériques pour les périphériques.



Système d'exploitation

Définition : (wikipedia)

ensemble de programmes responsables de la liaison entre les ressources matérielles d'un ordinateur et les applications informatiques de l'utilisateur (traitement de texte, jeux vidéo, etc). Il fournit aux programmes d'applications des points d'entrée génériques pour les périphériques.



Système d'exploitation et tâches

Processus :

c'est un programme en cours d'exécution. Un programme peut lancer une à plusieurs tâches à la fois. (tâche $\approx$ processus)

Il appartient au système d'exploitation de :

- dire quelle tâche passe après quelle autre (ordonnancement),
- synchroniser les tâches si elles collaborent,
- faire communiquer les tâches si besoin (ex : valeur d'une variable),
- détruire, commencer, suspendre les tâches.

Gestion des tâches

Taxonomie :

- Tâche immédiate : générées par une interruption sur le μp (en général par une E/S), elles doivent être traitées rapidement et sont toujours plus prioritaires que les tâches différées. Elles sont traitées par le système d'interruption (cf. diapo 43) sur lequel on ne peut pas intervenir.

- Tâche différée : générées par une application logicielle. Elles peuvent faire l'objet d'une programmation d'ordonnancement spécifique.

Gestion des tâches

Taxonomie :

- Tâche indépendante : l'exécution d'une telle tâche ne requiert pas l'intervention ou l'autorisation d'une autre tâche.
- Tâche inter-dépendante : une telle tâche a son exécution liée d'une manière ou d'une autre à une tâche tierce (autorisation à recevoir ou à envoyer, service en attente ou à rendre, etc.).

Gestion des tâches

Un processus se compose de différentes zones mémoires (segments) :

- segment de code (code segment) : contient les instructions codées par le programmeur (langage machine),
- segment de données (data segment) : contient les données modifiées, stockées et générées par le programme,
- segment de pile (stack segment) : contient des données temporaires (accessibles plus facilement),
- le tas (heap) : allocation dynamique à la suite du data segment. (non-systématique)

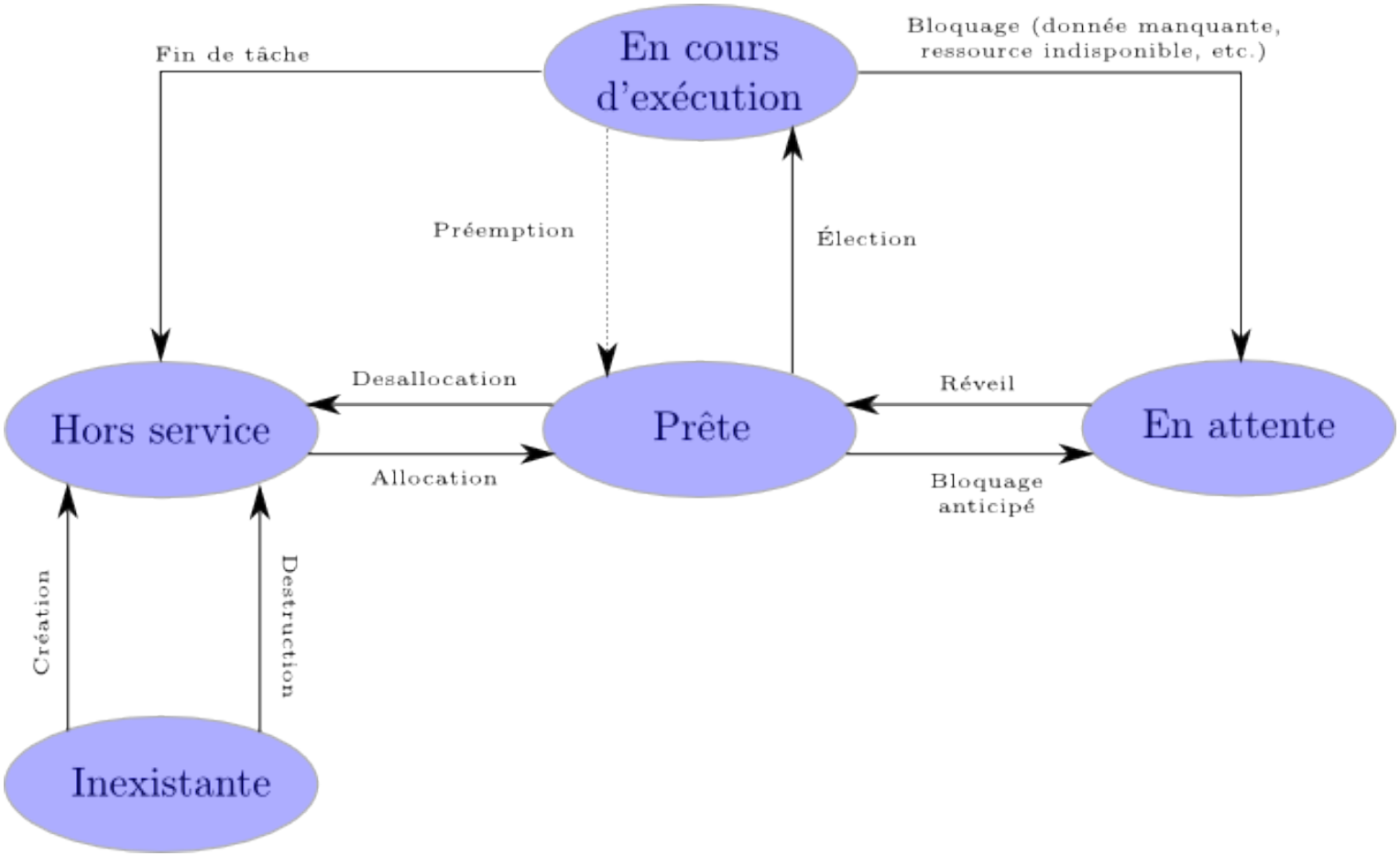
Gestion des tâches

Le contexte d'une tâche se définit comme l'ensemble des éléments permettant la poursuite de l'exécution à l'instant où une tâche est interrompue :

- 3 pointeurs vers les segments,
- nom (id),
- droits d'accès (ressources matérielles et logicielles accessibles),
- identificateur de l'état courant.
- identité de l'utilisateur (admin, user, SE) auquel est rattachée la tâche

Gestion des tâches

Cycle de vie d'une tâche :



Gestion des tâches

Les états :

- en cours d'exécution (courante) : une seule tâche peut être dans cet état. C'est elle qui accède à l'UC, "elle a la main".
- prête : elle demande à être exécutée.
- en attente : elle attend un événement, une ressource non-partageable ou un délai.
- hors service : elle existe et est présente en mémoire mais ne demande pas encore l'UC
- inexistante : non créée, non initialisée ou morte.

Gestion des tâches

Les transitions :

- exec $\rightarrow$ HS : la tâche vient de se terminer
- exec $\rightarrow$ prête : la tâche est interrompue par le SE (préemption)
- exec $\rightarrow$ attente : la tâche demande au SE d'être mise en attente, car elle a besoin de qqch qui n'est pas encore là.

Gestion des tâches

Les transitions :

- prête $\rightarrow$ HS : à la demande d'une autre tâche qui supervise.
- HS $\rightarrow$ prête : à la demande d'une autre tâche qui supervise.
- prête $\rightarrow$ attente : à la demande d'une autre tâche qui supervise.

Gestion des tâches

Les transitions :

- en attente $\rightarrow$ prête (réveil) : délai de mise en attente terminée OU événement survenu OU ressource libérée OU sur demande d'une autre tâche qui supervise.
- HS $\rightarrow$ inexistant : libération mémoire.
- inexistant $\rightarrow$ HS : allocation mémoire.

Gestion des tâches

Nota Bene :

- Une transition a lieu si
  - la tâche le demande,
  - une tâche supérieure qui supervise le demande,
  - le SE en décide ainsi (ordonnancement).
- Une tâche immédiate ne subit pas ces phénomènes, elle peut accéder à l'ordonnanceur mais ne peut être bloquée par ce dernier.
- Quand une tâche perd la main, tout son contexte doit être sauvegardé pour qu'une fois qu'elle la récupère, elle reprenne là où elle en était. Paradoxe : une commutation de tâches est coûteuse en temps, mais on ne peut pas faire de temps réel sans cet outil.

Qualificatifs de tâches

Tâches et fréquences d'activation :

- tâche périodique : elle est activée à intervalles de temps réguliers et connus du programmeur. Son exécution doit être achevée avant la fin de la période.
- tâche non-périodique :
  - tâche sporadique : un temps minimal a entre deux activations est défini et connu du programmeur. L'exécution doit être terminée avant a (contrainte forte)
  - tâche apériodique : la tâche apparaît de façon non-prévisible, on se contente de contraintes souples sur le temps de réponse moyen.

Descripteurs (éventuels) de tâches

Possibles attributs quantificatifs d'une tâche P_i :

- la période T_i (si tâche périodique),
- le temps de traitement maximal ou capacité C_i,
- la date d'activation ou de réveil r_i,
- le délai critique D_i (délai maximum admissible pour l'exécution de P_i),
- l'échéance $R_i = r_i + D_i$,
- la laxité $L_i = D_i - C_i$.

Descripteurs (éventuels) de tâches

- Pour une tâche périodique la k^{eme} date d'activation r_i^k vaut $r_i + k * T_i$.
- Quand $D_i = T_i$, on parle de tâche à échéance sur requête.
- Il va de soi que la logique veut que : $\forall i,\ 0 < C_i < D_i < T_i$.
- Connaître ces paramètres est quasiment indispensable pour faire la programmation temps réel.
- Une tâche est dite faisable si toutes ses instances peuvent se terminer avant leur échéances.

Conséquence sur l'UC

- facteur d'occupation de l'UC par la tâche P_i : $u_i = \frac{C_i}{T_i}$,
- facteur de charge de l'UC par la tâche P_i : $ch_i = \frac{C_i}{D_i}$,
- On définit la charge globale comme : $U = \sum_i u_i = \sum_i \frac{C_i}{T_i}$.

Interruptions

Il existe deux manières principales d'interagir avec l'environnement :

- par scrutation cyclique des périphériques (les périphériques sont interrogés à intervalles de temps réguliers sur la présence d'un événement à traiter),
- par système d'interruptions.

Interruptions

On ne peut espérer concevoir un système temps réel efficace sans utiliser d'interruptions. Elles permettent de réagir aux événements et donc d'être en phase avec l'environnement du système.
Il existe deux grandes catégories d'interruptions :

- matérielle : signal physique câblé sur le processeur. Un changement d'état du signal provoque une rupture de séquence sur le μp. Elle génère une tâche immédiate.
- logicielle (trap) : elle est générée par une application ou par le système.

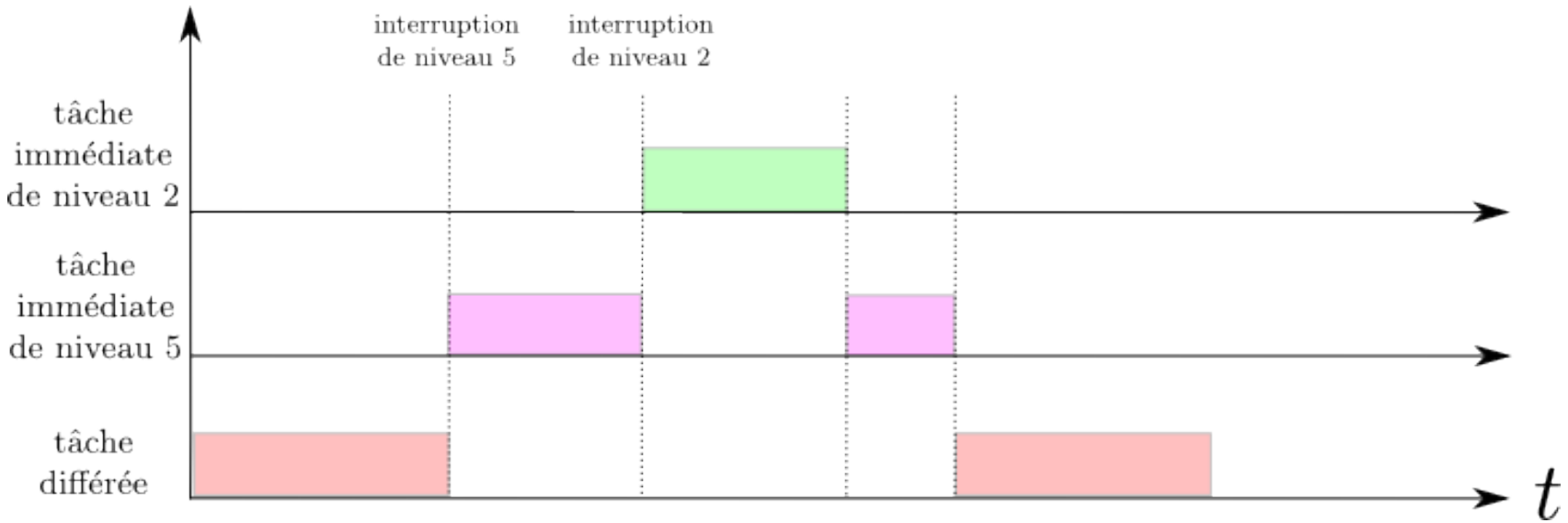
On distingue également les interruptions en fonction de l'événement auquel elles correspondent (E/S, horloge, défaut de page de mémoire).

Interruptions

- Le système d'interruption (ou gestionnaire d'interruption) se charge de faire commuter les tâches et d'appeler une routine éventuellement associée à l'interruption.
- Comme pour les tâches, il faut également gérer plusieurs niveaux de priorité pour les interruptions.
- Cette gestion s'opère à l'aide de masques applicables sur le code d'une interruption.

Interruptions

Exemple :



Interruptions

Attention à ne pas mélanger :

- une interruption matérielle engendre la création d'une tâche immédiate (et donc prioritaire par rapport à toutes celles dans les files d'attente de l'ordonnanceur). Cette tâche correspond souvent à une routine (petit programme en dur).
- une interruption logicielle provoque une commutation de tâches différées.
- une tâche différée peut demander à générer une interruption logicielle si elle supervise la tâche courante.

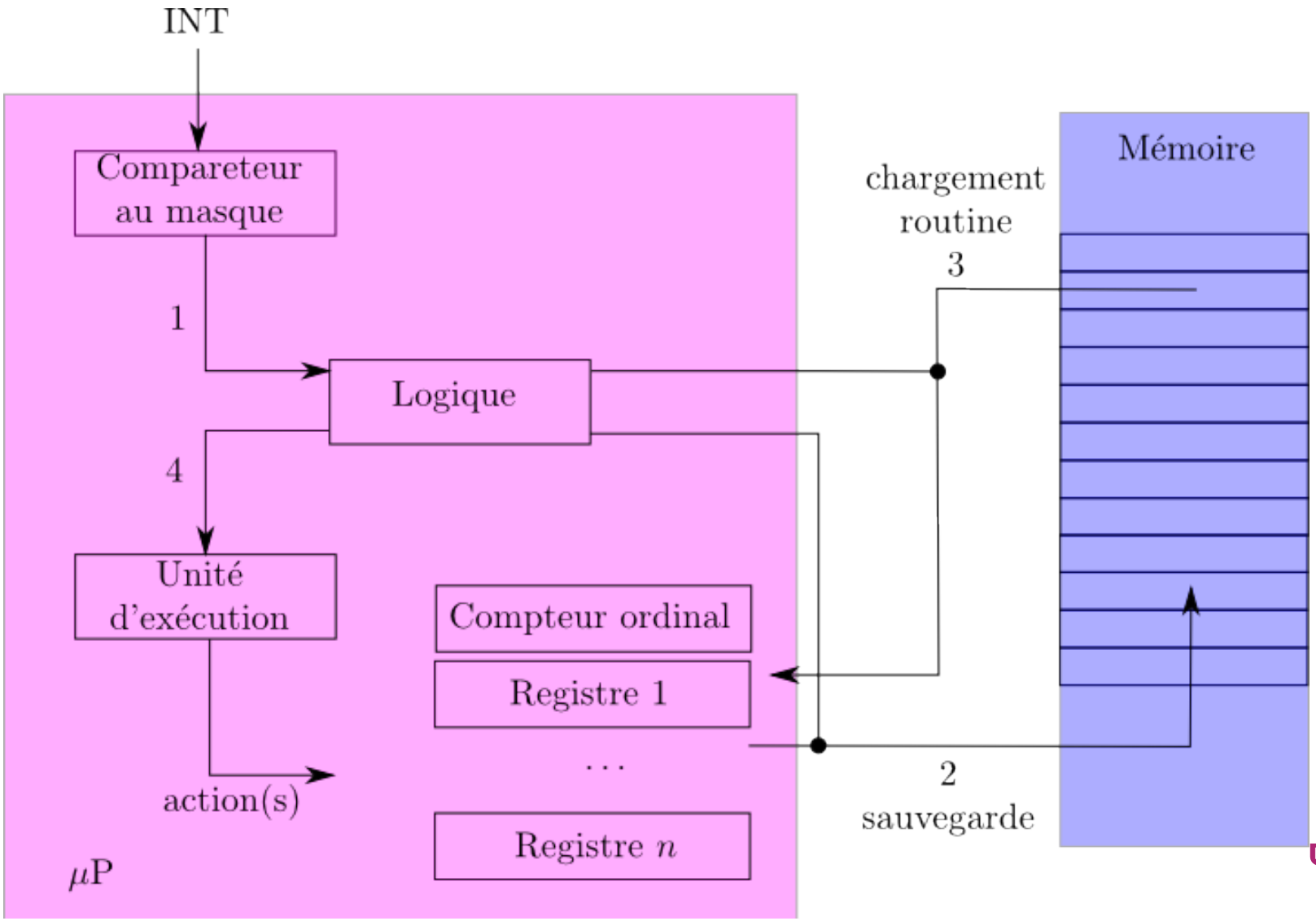
Interruptions

Séquence d'interruption :

- si non masquée, l'interruption arrive au μp. Une interruption ne peut être prise en compte qu'à la fin de l'instruction en cours (atomicité),
- le contexte de la tâche courante est sauvegardé,
- la routine associée à l'interruption est lancée,
- un acquitement est envoyé pour signifier la fin du traitement de l'interruption,
- si une autre interruption de priorité inférieure est en attente, elle est traîtée, sinon on appelle l'ordonnanceur pour traiter la prochaîne tâche différée et charger le nouveau contexte.

Interruptions

Séquence d'interruption :



Interruptions

Pour gérer les interruptions, il faut également être capable :

- de stocker les interruptions en file d'attente,
- de déterminer l'origine d'une interruption.

Interruptions

L'identification de la source de l'interruption peut être :

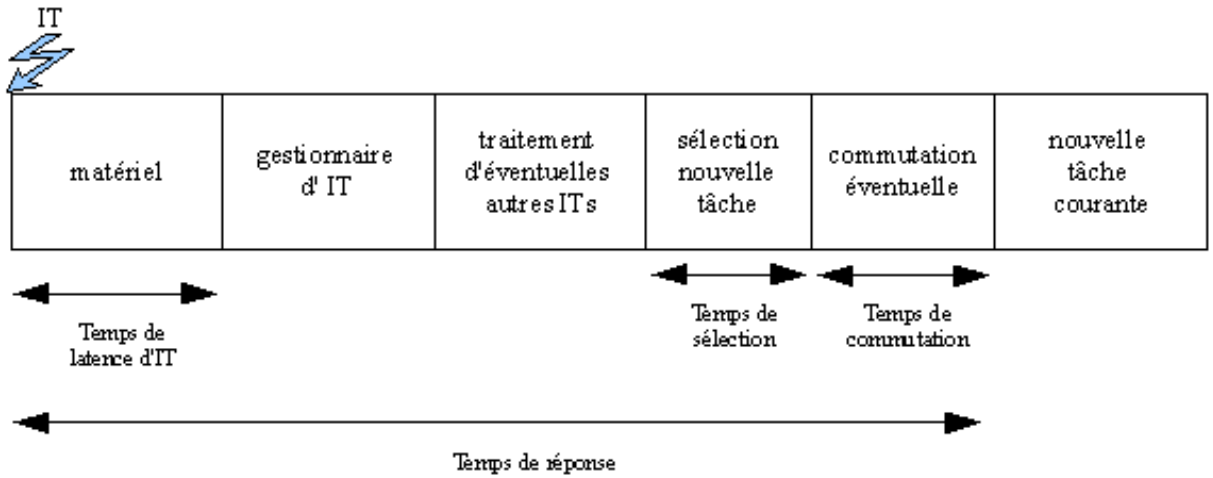
- directe : un niveau de priorité est directement associé à un périphérique (très rapide),
- par scrutation (polling), une requête est envoyée à tous les périphériques générant l'interruption reçue. Le "coupable" se fait alors connaître. (un même type d'intérruption partagé par plusieurs périphériques)
- par vectorisation : le périphérique envoie sur le bus de données un numéro de vecteur, traitée comme une adresse vers la routine de la part de l'UC. (besoin d'une table prédéfinie)

Interruptions

D'un point de vue temps réel il faut contrôler :

- le temps de commutation (temps moyen pris par le système pour commuter entre deux tâches),
- le temps de sélection (temps moyen pris par le système pour déterminer l'identité de la prochaine tâche)
- le temps de latence des interruptions (temps entre la réception de l'interruption et le déclenchement de la routine correspondante)
- le temps de latence d'entrée dans la tâche (somme de toutes les phases du traitement d'une interruption)
- le temps de réponse (temps de latence d'entrée dans la tâche mais avec éventuellement plusieurs interruptions au milieu)

Interruptions



Interruptions

Toujours d'un point de vue temps réel il est à noter que :

- on a besoin des interruptions,
- elles causent un aléa dans le temps de traitement d'une tâche,
- il est préférable de remonter un maximum d'interruption au niveau logiciel pour garder un contrôle dessus (et limiter l'aléa).

Interruptions

Exercice : contraintes temps réel et interruptions

Le partage de ressources critiques

- Il existe certaines ressources matérielles (imprimantes) ou logicielles (variables globales) qui ne doivent être allouées que à une seule tâche à la fois
- Solutions disponibles pour régler cette question :
  - masquer toutes les interruptions (lourd et possiblement gênant)
  - mettre un verrou : variable booléenne V (1 accès verrouillé, 0 accès autorisé) et une file d'attente $F(V)$ pour les tâches qui demandent le verrou quand celui-ci est déjà posé (pas d'attente active)
  - utiliser un sémaphore (généralisation du verrou à plusieurs points d'entrée dans la section)

Le partage de ressources critiques

Le sémaphore :

- variable entière S et file d'attente $F(S)$
- deux primitives : attendre/demander $P(S)$ et signaler/libérer $V(S)$

$P(S)$:
$S \leftarrow S - 1$
si $S \leq 0$ alors suspendre tâche appelante et l'insérer dans $F(S)$.

$V(S)$:
$S \leftarrow S + 1$
si $S \leq 0$ alors réveiller la tâche en tête de $F(S)$

Le partage de ressources critiques

Le sémaphore :

- si $S > 0$, S est le nombre de droit d'entrée en section critique (en général S vaut 1 au maximum mais cela peut être plus),
- si $S \leq 0$, S est le nombre de tâches en attente de la section critique.

Interbloquage (deadlock) :

tâche 1		tâche 2
$P(S_1)$	1	
	2	$P(S_2)$
utiliser ressource 1	3	
	4	utiliser ressource 2
$P(S_2)$	5	
	6	$P(S_1)$
bloquée		**bloquée**

Le partage de ressources critiques

Exercice : partage de voies dans un système industriel

Synchronisation des tâches

Certaines tâches sont inter-dépendantes, il faut donc pouvoir s'assurer qu'une tâche A est à un tel point d'avancement avant qu'une tâche B exécute telle instruction ou qu'un événement ait eu lieu.
plusieurs types :

- directe : les tâches en relation connaissent leurs "identités" respectives,
- indirecte : pas besoin d'identification .

Synchronisation des tâches

Méthode directe : tâche B supervise tâche A.

- deux variables booléennes : $ETAT$ (0 = bloquée, 1 = éveillée), $FANION$ (1 = signal d'éveil envoyé)
- deux primitives : $BLOQUER\,()$ et $EVEILLER\,(A)$

$BLOQUER\,()$ (appelée par A) :

Si $FANION = 1$ alors $FANION \leftarrow 0$
Sinon appel système pour passer en attente, $ETAT \leftarrow 0$

$EVEILLER\,(A)$ (appelée par B) :

Si $ETAT = 0$ alors appel système pour réveiller A, $ETAT \leftarrow 1$
Sinon $FANION \leftarrow 1$

Synchronisation des tâches

Méthode par sémaphore (indirecte) : Sémaphore S initialisée à 0, le code de la tâche A contient un $P(S)$ et celui de la tâche B un $V(S) \Rightarrow$ A est bloquée tant que B n'a pas fait toutes les instructions précédant le $V(S)$.

Exercice :
1) faire en sorte qu'une tâche A s'exécute si les tâches B OU C ont fini une certaine instruction.
2) faire en sorte qu'une tâche A s'exécute si les tâches B ET C ont fini une certaine instruction.

Synchronisation des tâches

Synchronisation par événements :

- exemple classique : une tâche attend un traitement effectué par un périphérique. Une fois terminé, le périphérique lève une interruption, la routine signale l'événement, la tâche en attente peut continuer.
- l'événement peut être global (concerne toutes les tâches, ou un groupe de tâche) ou local (une tâche en particulier)
- l'événement peut être identifié par un mot de code (id)
- une variable booléenne Ev ($0 =$ non arrivé, $1 =$ arrivé)
- une file d'attente $F(Ev)$
- trois primitives $SET()$, $RESET()$ et $WAIT()$

Synchronisation des tâches

SET () (appelé par routine ou tâche) :

$Ev \leftarrow 1$
répéter
  si $F(Ev) \neq \emptyset$ **alors**
    appel système pour débloquer le processus en attente
  fin si
jusqu'à $F(Ev) = \emptyset$

RESET () (appelé par une tâche satisfaite) :

$Ev \leftarrow 0$
vider $F(Ev)$

Synchronisation des tâches

WAIT () (appelé par la ou les tâche(s) ayant besoin du signal) :

si $Ev = 0$ **alors**
insérer la tâche dans $F(Ev)$
fin si

Remarques :

- Attention à ne pas effacer un signal avant de l'avoir pris en compte

Synchronisation entre tâches

Le rendez-vous à N points d'entrées :

- N tâches (au max) peuvent demander un rdv,
- besoin d'un compteur de tâches en attente CPT
- besoin d'un système de masquage pour rendre le code gérant le rdv ininterruptible
- besoin de signaux pour gérer la file d'attente des tâches

Synchronisation entre tâches

Le rendez-vous à N points d'entrées (code générique) :
RDV_init() :

$mutex \leftarrow 1$
$CPT \leftarrow 0$ (nbr de tâches arrivées)

Synchronisation entre tâches

Le rendez-vous à N points d'entrées (code générique) :

RDV_proc() :

$P(mutex)$
$CPT \leftarrow CPT + 1$
si $CPT < N$ **alors**
  $V(mutex)$
  wait()
sinon
  service éventuel
  set(all)(réveil des tâches par diffusion)
  $CPT \leftarrow 0$
  $V(mutex)$
fin si

Synchronisation entre tâches

Le rendez-vous à N points d'entrées, remarques :

- Plusieurs exemples sur le web notamment en langage Java,
- pas forcément besoin de connaître l'identité des tâches entrantes.

Communication entre tâches

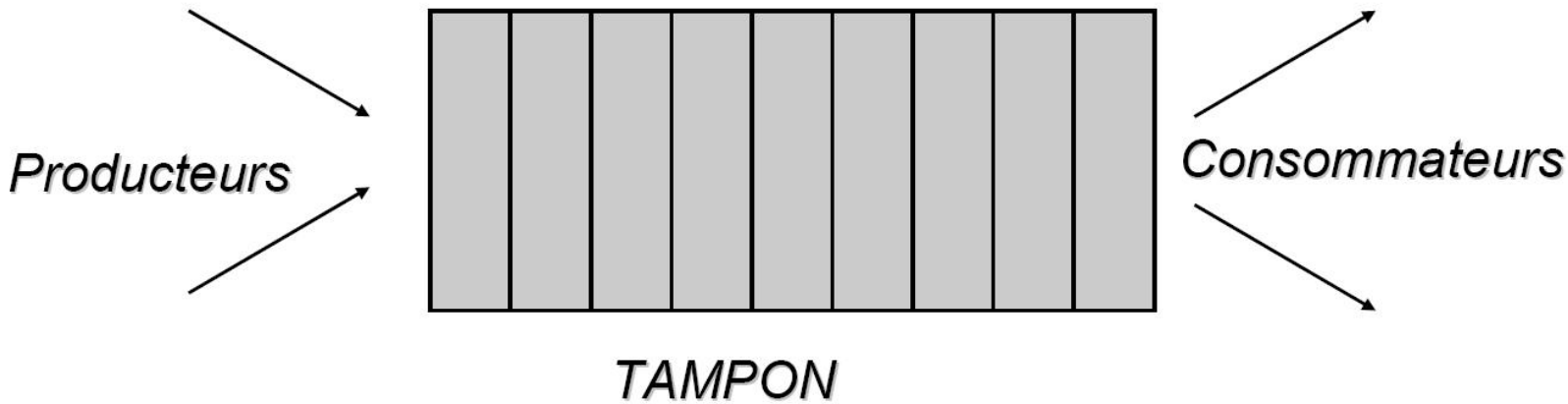
Communication = synchronisation + transmission de données.
Techniques les plus répandues :

- une variable globale (avec éventuellement un *timestamp*),
- le schéma producteur-consommateur,
- la boîte à lettres,
- le tube.

Communication entre tâches

Le schéma producteur-consommateur :

- un tampon est alloué et sert à déposer des messages fournis par des producteurs.
- les consommateurs accèdent en lecture au tampon et libèrent un emplacement par lecture.
- le tampon a une taille limitée de N messages.
- dans un tube, pas de limite sur N.



Communication entre tâches

La boîte à lettres : idem mais tous les messages sont au même format.

- 1 sémaphore *place* correspond à des jetons de places disponibles dans le tampon (init à N),
- 1 sémaphore *mail* correspond à des jetons de messages disponibles dans le tampon (init à 0),
- deux primitives : *ENVOYER*() et *RECEVOIR*(),
- le dépôt de message doit être non interruptible,
- la primitive *RECEVOIR*() est bloquante.

Communication entre tâches

ENVOYER() :

P (*place*)
déposer(message,destinataire)
V (*mail*)

RECEVOIR() :

P (*mail*)
retirer(message)
V (*place*)

Communication entre tâches

Exercice : approvisionnement automatisé d'un site industriel

Ordonnancement

Définition :

L'ordonnacement (*scheduling*) est la politique selon laquelle le système d'exploitation organise le partage de l'UC entre différentes tâches. Il doit faire en sorte que toutes les tâches soient faisables.

Ordonnanceur :

ensembles de routines du SE qui mettent en oeuvre la politique d'ordonnancement.

Ordonnancabilité d'un ensemble de tâches :

Il existe au moins un algorithme faisant en sorte que toutes les tâches respectent leur contraintes.

Ordonnancement

Un ordonnanceur A est optimal si $\forall \Gamma = \{P_1, ..., P_n\}$ un ensemble de tâches faisables par un autre ordonnanceur B, l'ordonnanceur A peut également produire un ordonnancement valide pour Γ.

On distingue les ordonnancements :

- non-préemptifs : n'interviennent que si la tâche courant se finit ou demande à être mise en attente.
- préemptifs : sont capables de retirer l'UC à une tâche pour la donner à une autre.

Ordonnancement

On distingue également les ordonnancements :

- off-line : la séquence d'ordonnancement est déterminée à l'avance,
- on-line : la séquence d'ordonnancement est décidée durant l'exécution du système.

On parle enfin d'ordonnancements :

- statiques : les paramètres des tâches guidant l'ordonnancement ne peuvent pas évoluer,
- dynamiques : les paramètres des tâches guidant l'ordonnancement peuvent évoluer pendant l'exécution du système. Une même tâche peut donc être ordonnancée différemment à deux instants différents.

Ordonnancement

Commutation de tâches :

Il s'agit du fait de faire passer la tâche courante dans un autre état et de lui substituer une autre tâche prête.

remarque :

- Dans la grande majorité des cas, une commutation de tâche implique un changement de contexte (le contexte actuel est sauvegardé et le nouveau chargé).
- Si la tâche courante est terminée, alors seul le nouveau contexte est chargé, l'ancien est détruit.

Ordonnancement

Commutation de tâches en 3 étapes :

- 1 : arrêt de la tâche courante,
- 2 : élection par l'ordonnanceur de la prochaine tâche à exécuter,
- 3 : si ancienne tâche non terminée, alors changement de contexte.

Remarque :

La plupart des systèmes mettent en place une **tâche de fond** quand toutes les tâches "réelles" sont terminées. Cela permet de commuter avec "quelque chose".

Ordonnancement

Politique classique n°1 :

First In First Out (**FIFO**), premier arrivé premier servi.

- algorithme non-préemtif.
- tâches placées en file d'attente.
- nouvelle arrivante placée en queue de file.
- + : simple à mettre en oeuvre.
- - : inadapté au temps réel (respect des contraintes de temps non garantissable).

Ordonnancement

Politique classique n°2 :

Tourniquet (**round robin**), partage tour à tour de l'UC, changement à intervalle de temps constant (*quantum*).

- algorithme non-préemtif.
- fonctionnement en FIFO avec commutation à chaque *quantum*.
- nouvelle arrivante placée en queue de file.
- + : assurance d'un partage de l'UC.
- - : ajustement du *quantum* très délicat, pas de possibilité de prendre en compte une tâche urgente.

Ordonnancement

Politique classique n°3 :

Ordonnancement par **priorité**, un niveau de priorité est accordé à chaque tâche, les plus prioritaires pouvant préempter les moins prioritaires.

- algorithme préemtif.
- fonctionnement en FIFO ou tourniquet pour tâches de même priorité (**politique mixte**).
- + : traitement de tâches urgentes possible (solution retenue pour les systèmes temps réel).
- - : assignation des priorités délicate, risque de "famine" pour les tâches peu prioritaires.

Ordonnancement

Exemple : trois tâches T_1, T_2 et T_3 arrivées dans cet ordre, sont traitées par FIFO. Leurs durées sont respectivement de 6, 3 et 4 unités de temps.
L'utilisation de l'UC est alors la suivante :

T_1	T_1	T_1	T_1	T_1	T_1	T_2	T_2	T_2	T_3	T_3	T_3	T_3

Ordonnancement

Exemple : trois tâches T_1, T_2 et T_3 arrivées dans cet ordre, sont traitées par round robin. Leurs durées sont respectivement de 6, 3 et 4 unités de temps. (*quantum* = 1 unité)
L'utilisation de l'UC est alors la suivante :

T_1	T_2	T_3	T_1	T_2	T_3	T_1	T_2	T_3	T_1	T_3	T_1	T_1

Ordonnancement

Exemple : trois tâches T_1, T_2 et T_3 arrivées dans cet ordre, sont traitées par priorités. Leurs durées sont respectivement de 6, 3 et 4 unités de temps et leurs priorités 1, 2 et 1. (priorité 2 $>$ priorité 1)
L'utilisation de l'UC est alors la suivante :

T_2	T_2	T_2	T_1	T_1	T_1	T_1	T_1	T_1	T_3	T_3	T_3	T_3

Ordonnancement

Exercice : trois tâches T_1, T_2 et T_3 arrivées dans cet ordre, sont traitées par priorités et round robin si priorités égales. Leurs durées sont respectivement de 6, 3 et 4 unités de temps et leurs priorités 1, 2 et 1. (priorité 2 $>$ priorité 1)
Quelle est l'utilisation de l'UC correspondante ?

T_2	.	.	.	.	.	.	.	.	.	.	.	.

Ordonnancement

Pour des aspects temps réel, il est intéressant de pouvoir évaluer les politiques d'ordonnancement grâce à un test d'ordonnançabilité :

- On a déjà vu, que pour des raisons évidentes la charge de l'UC $U = \sum_i \frac{C_i}{T_i}$ doit être inférieure à 1 pour un système mono-processeur.
- U dépend évidemment des caractéristiques de l'ensemble de tâches $\Gamma = \{P_1, ..., P_n\}$ à ordonnancer.
- Pour certains algorithmes d'ordonnancement (algo A par exemple), il est possible de déterminer une valeur $U_{ub}(\Gamma, A)$ en dessas de laquelle Γ sera ordonnançable par A à coup sûr.
- L'expression de U_{ub} est obtenue à l'aide d'une analyse mathématique de l'algorithme A (cette expression varie donc d'un algo à l'autre).

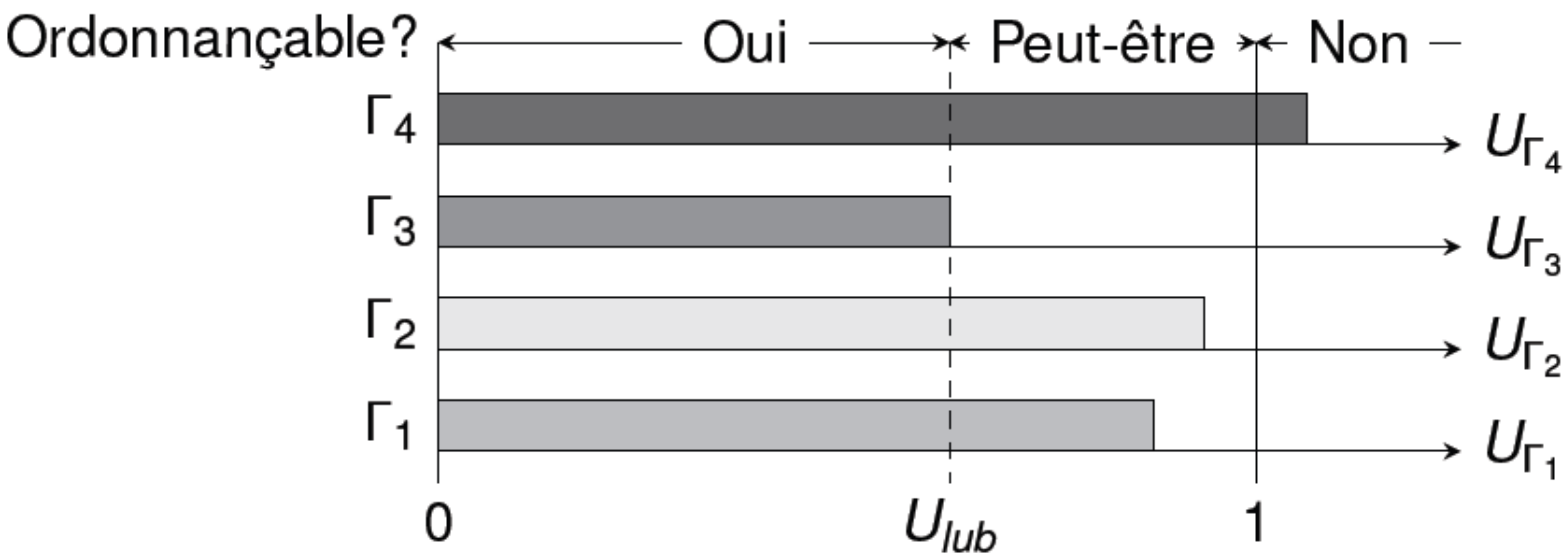
Ordonnancement

- Quand on teste l'ordonnançabilité de A, on s'intéresse particulièrement à la valeur : $U_{lub}(A) = \min_{\Gamma} U_{ub}(\Gamma, A)$. Cette valeur est également obtensible par analyse mathématique de A.
- Ainsi, quand on veut savoir si un ensemble Γ de tâches sera ordonnançable par A, on vérifie que :

$$U(\Gamma) \leq U_{lub}(A) \tag{1}$$

- Il s'agit d'une condition suffisante mais pas nécessaire.
- *ub* signifie *upper bound* (borne supérieure) et *lub least upper bound* (borne supérieure minimale).

Ordonnancement



Ordonnancement

Sur quel intervalle de temps étudier l'ordonnançabilité ?

- tâches uniquement périodiques et synchrone : $[0, ppcm(T_1, ..., T_n)]$,
- tâches uniquement périodiques mais asynchrone : $[\min r_i, \max(r_i) + 2ppcm(T_1, ..., T_n)]$.
- tâches périodiques asynchrones (indexées par i) et apériodiques (indexées par j) : $[\min r_i, \max(R_i) + 2ppcm(T_1, ..., T_n)]$.

Ordonnancement

Politique analysable n°1 : Rate Monotonic Analysis (RMA)

Hypothèses de départ fortes :

- les tâches à ordonnancer sont périodiques et sont à l'état "prêt" en début de chaque période,
- $\forall i$, $D_i = T_i$,
- $\forall i$, C_i est connu
- les tâches sont indépendantes,
- les tâches ne demandent pas elles-mêmes à être mises en attente,
- l'algorithme est préemtif et les temps de commutation et d'ordonnancement sont négligeables par rapport aux C_i.

Ordonnancement

Politique analysable n°1 : Rate Monotonic Analysis (RMA)

Principe : ordonnancement par priorité : $prio_i \propto 1/T_i$.

- algorithme on-line statique,
- $+$: l'analyse des paramètres de tâches permet d'assurer le respect de toutes les contraintes de temps, la condition suffisante est $U(\Gamma) \leq U_{ub}(RMA, \Gamma) = n\left(2^{\frac{1}{n}} - 1\right)$ où n est le nombre de tâches à ordonnancer. On note que $U_{lub}(RMA) = \lim_{n \to \infty} n\left(2^{\frac{1}{n}} - 1\right) = ln(2) \approx 0.69$.
- - : hypothèses trop contraignantes pour de nombreux systèmes, CS imposant souvent de surdimensionner l'UC.

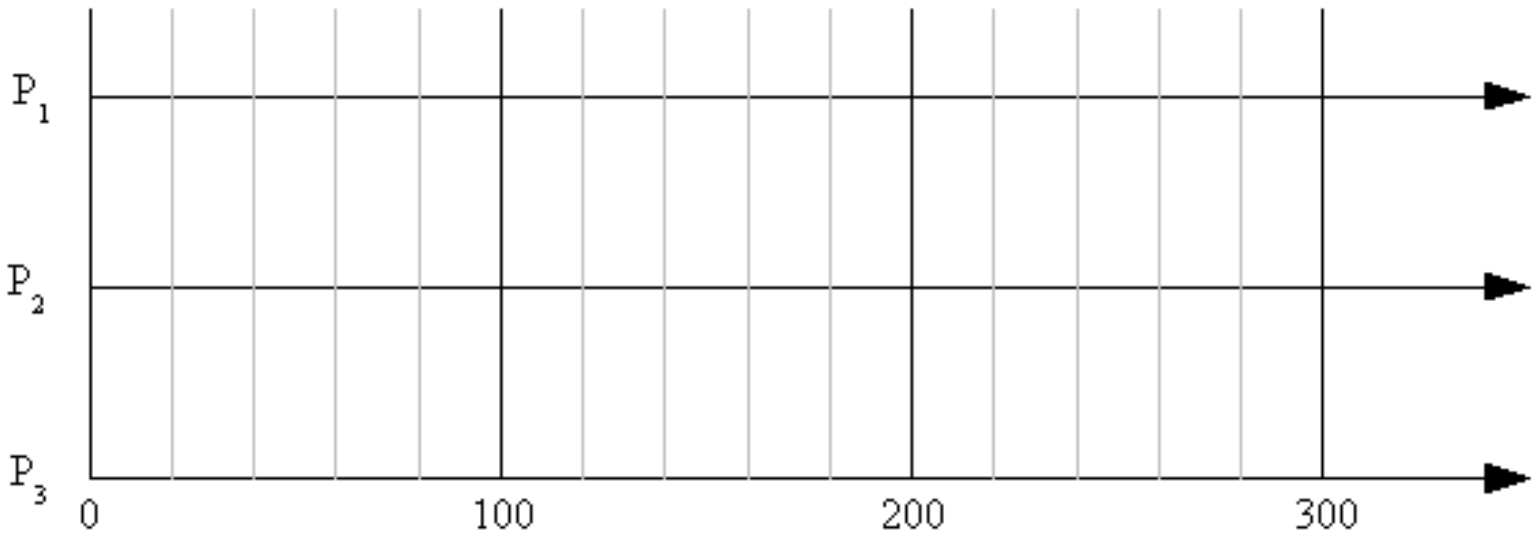
Ordonnancement

Politique analysable n°1 : Rate Monotonic Analysis (RMA)

Exemple : trois tâches P_1, P_2 et P_3. On a $C_1 = 20$, $T_1 = 100$, $C_2 = 40$, $T_2 = 150$, $C_3 = 100$ et $T_3 = 300$.

Questions :

- Ces tâches sont-elles ordonnançables à coup sûr par RMA ?
- Tracer le chronogramme des tâches



- Si $T_3 = 250$, les tâches sont-elles toujours ordonnançables à coup sûr par RMA ? Sont-elles ordonnançables en pratique ?

Ordonnancement

Théorème de la zone critique : énoncé

Soit un ensemble Γ de tâches périodiques indépendantes et à priorités variables. Si la tâche P_i respecte sa première échéance R_i alors que toutes les tâches de priorités supérieures appartenant à Γ sont prêtes, alors P_i respectera toutes ses échéances futures.

- principe facile à comprendre : si la tâche est faisable dans le pire des cas, elle sera faisable par la suite
- $+$: permet de tester l'ordonnançabilité par des algorithmes tels que RMA avec des contraintes moindres que celle de la charge optimale U_{lub}.
  Si toutes les tâches à ordonnancer sont initialement prêtes au même instant, c'est une condition nécessaire et suffisante.
- - : les hypothèses restent fortes.

Ordonnancement

Théorème de la zone critique : mise en oeuvre

- On commence par ordonner les tâches dans l'ordre décroissant de leurs priorités : P_1 est la tâche de priorité la plus forte, P_2 la tâche de priorité juste inférieure à P_1, etc. jusqu'à P_n la tâche de priorité la plus faible
- Soit $W_i(t) = \sum_{j=1}^{i} C_j \left\lceil \frac{t}{T_j} \right\rceil$.
- $\left\lceil \frac{t}{T_j} \right\rceil$ représente le nombre de fois que la tâche P_j est activée dans l'intervalle $[0, t]$.
- $C_j \left\lceil \frac{t}{T_j} \right\rceil$ représente la demande en consommation de la tâche P_j dans l'intervalle $[0, t]$.
- $W_i(t)$ représente la demande cumulée en UC de toutes les tâches de priorités plus fortes que celle de P_i dans $[0, t]$.

Ordonnancement

Théorème de la zone critique : mise en oeuvre

- On remarque que $W_i(t) = C_i + \sum_{j=1}^{i-1} C_j \left\lceil \frac{t}{T_j} \right\rceil$, autrement dit, la quantité $\sum_{j=1}^{i-1} C_j \left\lceil \frac{t}{T_j} \right\rceil$ représente la consommation supplémentaire (le retard) imposée à P_i par les tâches de priorités supérieures.
- Si il existe t tel que $W_i(t) = t$, alors le temps processeur a pu être alloué à P_i au bout de t.
- Pour étudier l'ordonnançabilité, il suffit donc de trouver pour toute tâche P_i un t tel que $W_i(t) = t$ et que évidemment $t \leq D_i$.

Ordonnancement

Théorème de la zone critique : mise en oeuvre

- Pour résoudre les n équations $W_i(t) = t$, on utilise une astuce d'analyse numérique :
  - On part de $t_0 = \sum_{j=1}^{i} C_j$ et on calcule $W_i(t_0)$,
  - Si $W_i(t_0) = t_0$, on a fini
  - Sinon on prend $t_1 = W_i(t_0)$ et on calcule $W_i(t_1)$,
  - On itère le processus jusqu'à tomber sur une bonne valeur de t_k ou que l'on dépasse l'échéance D_i.

Ordonnancement

Théorème de la zone critique : exemple

Reprendre l'énoncé de l'exemple avec RMA mais avec $T_3 = 250$.

- Les tâches sont-elles ordonnançables à coup sûr par RMA ?
- Appliquer le test en zone critique. Conclure sur l'ordonnançabilité.

Ordonnancement

Politique analysable n°2 : Deadline Monotonic Analysis (DMA)

Hypothèses de départ fortes (similaires à RMA sauf pour la deadline) :

- les tâches à ordonnancer sont périodiques et sont à l'état "prêt" en début de chaque période,
- $\forall i$, $D_i < T_i$ (ou $D_i > T_i$),
- $\forall i$, C_i est connu
- les tâches sont indépendantes,
- les tâches ne demandent pas elles-mêmes à être mises en attente,
- l'algorithme est préemtif et les temps de commutation et d'ordonnancement sont négligeables par rapport aux C_i.

Ordonnancement

Politique analysable n°2 : Deadline Monotonic Analysis (DMA)

Principe : ordonnancement par priorité : $prio_i \propto 1/D_i$.

- algorithme on-line statique,
- $+$: l'analyse des paramètres de tâches permet d'assurer le respect de toutes les contraintes de temps, la condition suffisante est $\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n\left(2^{\frac{1}{n}} - 1\right)$ où n est le nombre de tâches à ordonnancer.
- - : hypothèses souvent encore trop contraignantes pour de nombreux systèmes, CS surdimensionnant l'UC (pire que RMA).

Ordonnancement

Politique analysable n°2 : Deadline Monotonic Analysis (DMA)

- On peut également utiliser le test en zone critique pour DMA avec $W_i(t) = \sum_{j=1}^{i} C_j \left\lceil \frac{t}{T_j} \right\rceil$.
- La différence avec RMA est que l'ordre i des tâches peut être différent car les priorités sont calculées en fonction des échéances D_i.
- Les valeurs des t_0 trouvées doivent être comparées aux D_i et non aux T_i.

Ordonnancement

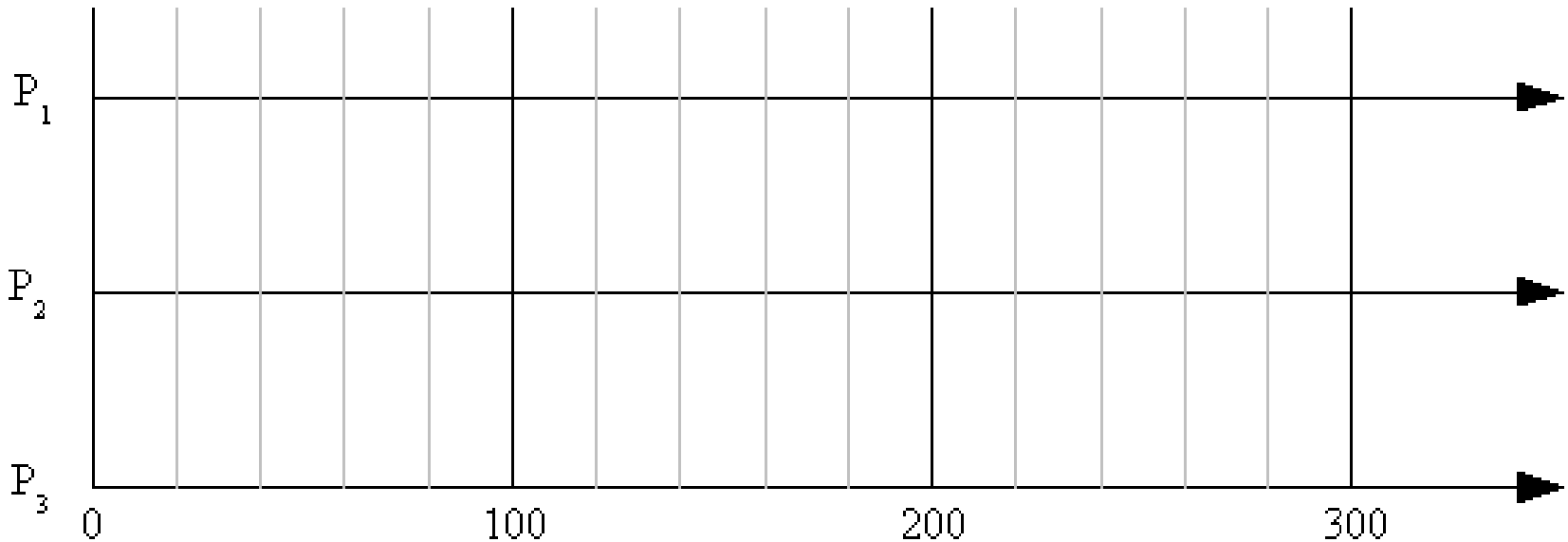
Politique analysable n°2 : Deadline Monotonic Analysis (DMA)

Exemple : Reprendre l'énoncé de l'exemple avec RMA mais avec $D_1 = 100$, $D_2 = 50$ et $D_3 = 300$.

- Les tâches sont-elles ordonnançables à coup sûr par DMA ? (CS de DMA)
- Appliquer le test en zone critique (faire attention aux priorités des tâches). Conclure sur l'ordonnançabilité.
- Remplir le chronogramme.

Ordonnancement

Politique analysable n°2 : Deadline Monotonic Analysis (DMA)



Ordonnancement

Politique analysable n°3 : Earliest Deadline First (EDF)

Hypothèses de départ fortes (similaires à RMA sauf pour la deadline) :

- les tâches à ordonnancer sont périodiques et sont à l'état "prêt" en début de chaque période,
- $\forall i$, D_i a priori quelconque,
- $\forall i$, C_i est connu
- les tâches sont indépendantes,
- les tâches ne demandent pas elles-mêmes à être mises en attente,
- l'algorithme est préemtif et les temps de commutation et d'ordonnancement sont négligeables par rapport aux C_i.

Ordonnancement

Politique analysable n°3 : Earliest Deadline First (EDF)

Principe : ordonnancement par priorité inversement proportionnelle à $D_i(t) = D_i - (t - r_i)$, le temps restant avant la prochaine échéance. En cas d'égalité, EDF laisse la tâche courante s'exécuter.

- algorithme on-line dynamique,
- $+$: l'analyse des paramètres de tâches permet d'assurer le respect de toutes les contraintes de temps, le test est :
  - $U(\Gamma) \leq 1$ si $\forall i, D_i \geq T_i$ (CNS),
  - $\sum_{i=1}^{n} \frac{C_i}{\min(D_i, T_i)} \leq 1$ sinon (CS).

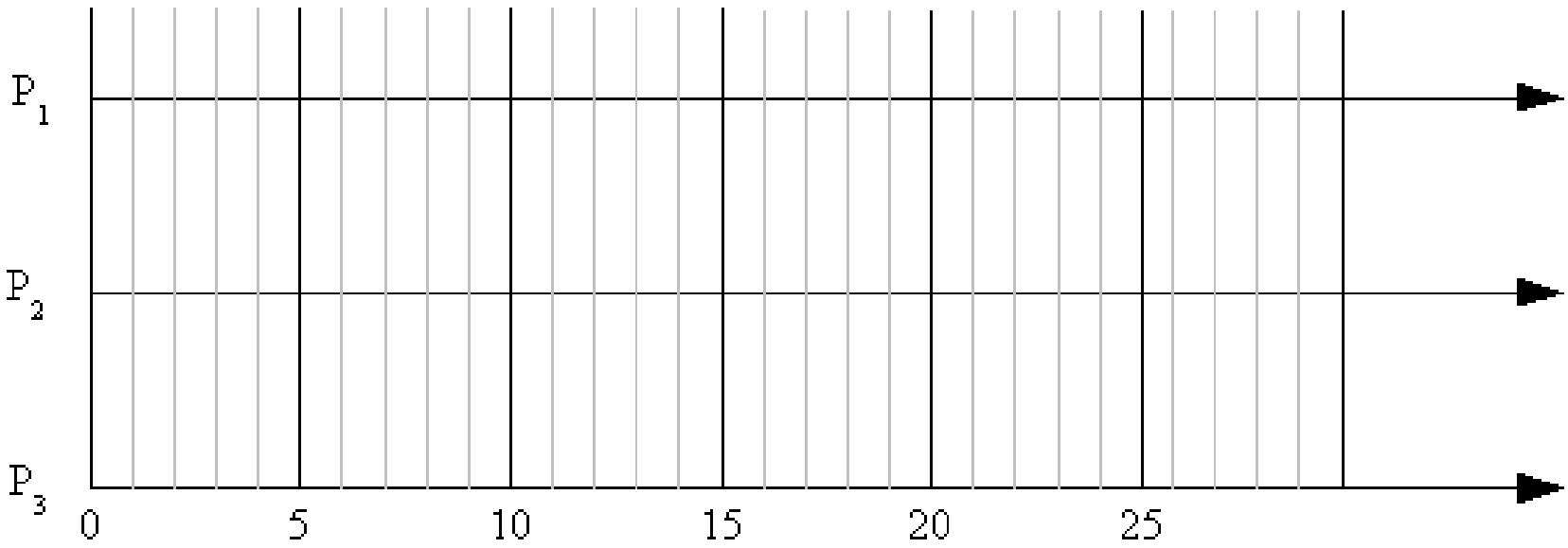
  Algorithme optimal (si un algo peut ordonnancer Γ alors EDF aussi).
- - : hypothèses toujours contraignantes, très sensible à la surcharge.

Ordonnancement

Politique analysable n°3 : Earliest Deadline First (EDF)

Exemple : soient trois tâches P_1, P_2 et P_3 avec $T_1 = 6$, $T_2 = 8$, $T_3 = 12$, $C_1 = 2$, $C_2 = 2$, $C_3 = 3$ et $\forall i$, $D_i = T_i$.

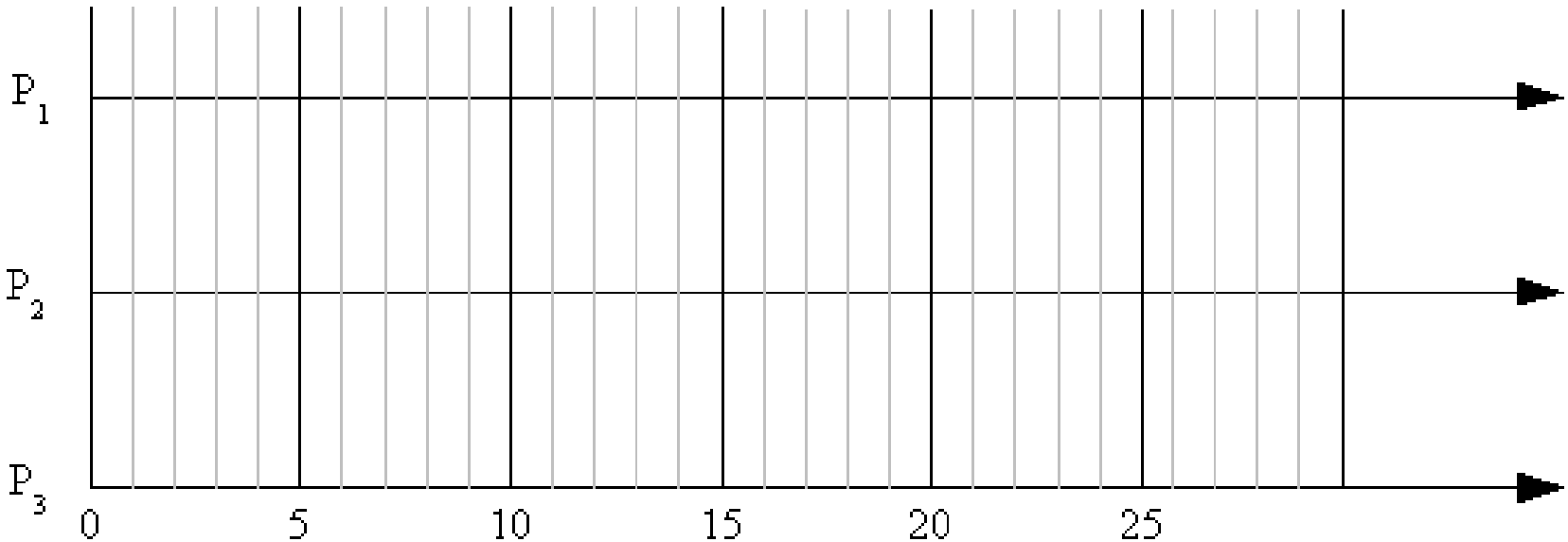
- Les tâches sont-elles ordonnançables à coup sûr par EDF ? (CNS de EDF)
- Remplir le chronogramme.



Ordonnancement

Politique analysable n°3 : Earliest Deadline First (EDF)

- Remplir le chronogramme en utilisant RMA afin de comparer.



Ordonnancement

Politique analysable n°4 : Least Laxity First (LLF)

Hypothèses de départ similaires à EDF :

- les tâches à ordonnancer sont périodiques et sont à l'état "prêt" en début de chaque période,
- $\forall i$, D_i a priori quelconque,
- $\forall i$, C_i est connu
- les tâches sont indépendantes,
- les tâches ne demandent pas elles-mêmes à être mises en attente,
- l'algorithme est préemtif et les temps de commutation et d'ordonnancement sont négligeables par rapport aux C_i.

Ordonnancement

Politique analysable n°4 : Least Laxity First (LLF)

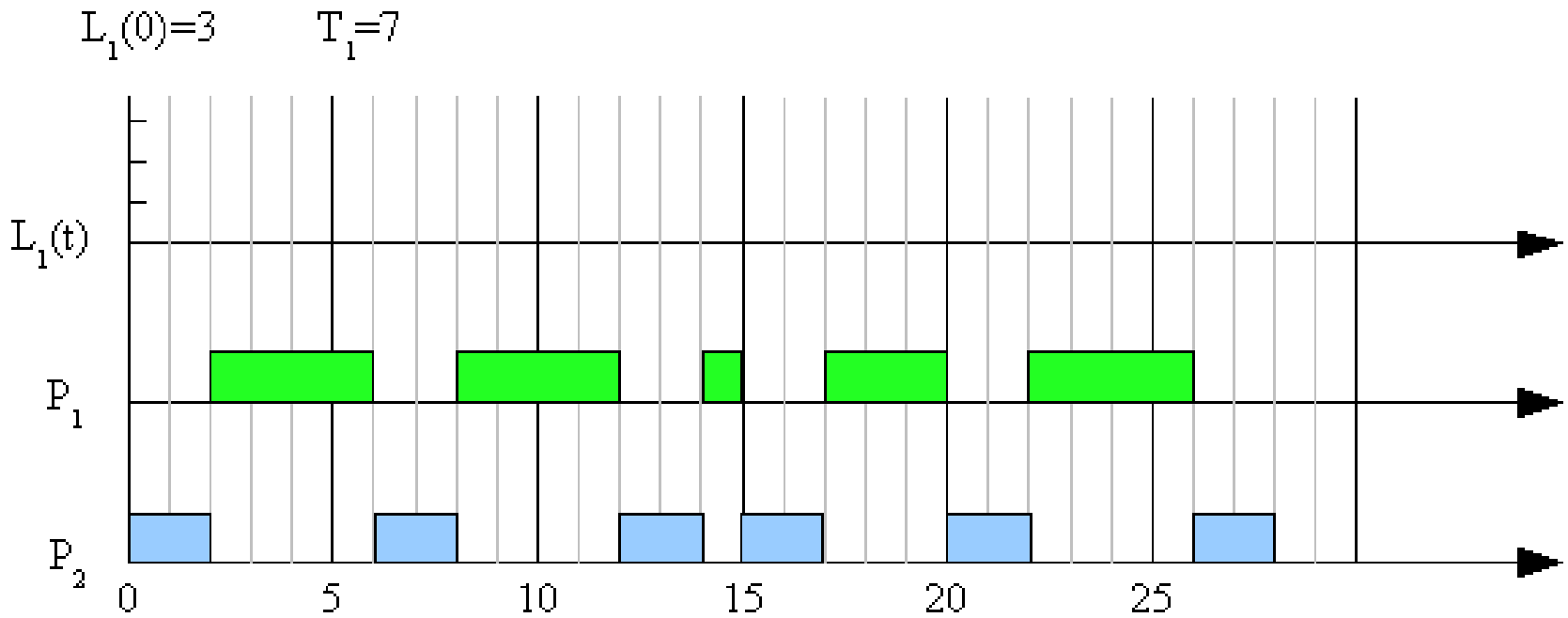
Principe : ordonnancement par priorité : $prio_i$ fonction décroissante de la laxité instantanée $L_i(t) = D_i(t) - C_i(t)$ avec $C_i(t)$ la charge restante. $C_i(t) = C_i -$ « temps d'exécution déjà effectué par P_i ».
En cas d'égalité, LLF laisse la tâche courante s'exécuter.
Caractéristiques semblables à EDF :

- algorithme on-line dynamique,
- $+$: l'analyse des paramètres de tâches permet d'assurer le respect de toutes les contraintes de temps, le test de faisabilité est identique à celui de EDF. Algorithme optimal (si un algo peut ordonnancer Γ alors LLF aussi).
- - : hypothèses toujours contraignantes, très sensible à la surcharge.

Ordonnancement

Politique analysable n°4 : Least Laxity First (LLF)

Exemple de laxité instantannée : remplir le chronogramme suivant

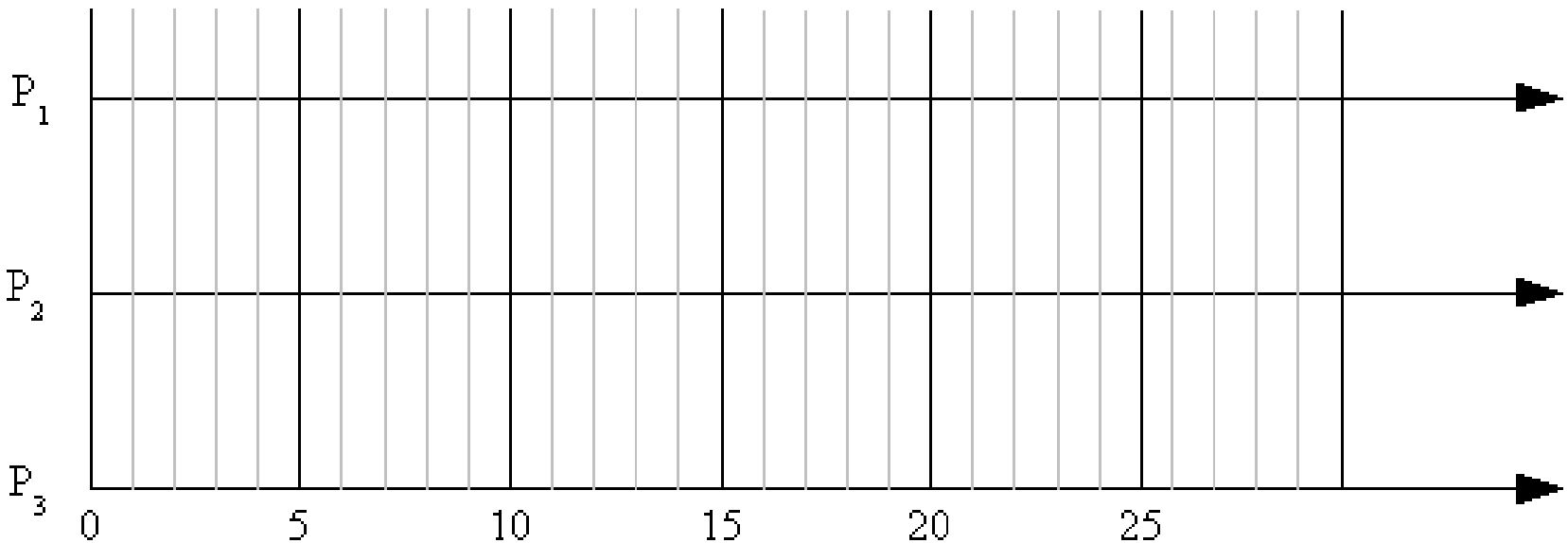


Ordonnancement

Politique analysable n°4 :Least Laxity First (LLF)

Exemple d'ordonnancement par LLF : Reprendre l'énoncé de EDF.

- Les tâches sont-elles ordonnançables à coup sûr par LLF ? (CNS de LLF)
- Remplir le chronogramme et comparer à EDF et RMA.



Ordonnancement

Ordonnacement de tâches dépendantes

Exclusion mutuelle : problème d'inversion des priorités

- Soient trois tâches P_1, P_2 et P_3 de priorités décroissantes.
- P_1 et P_3 partagent une ressource critique protégée par un sémaphore.
- P_3 prend le sémaphore puis est préemptée par P_1 qui demande le sémaphore et se trouve donc bloquée.
- P_3 devient en quelque sorte plus prioritaire que P_1.
- En outre, si P_2 préempte P_3 il est compliqué prévoir le temps d'attente de P_1.

Ordonnancement

Ordonnacement de tâches dépendantes

Exclusion mutuelle : problème d'inversion des priorités - solutions

- Empêcher la préemption d'une tâche en section critique.
- Héritage de priorité simple : si une tâche P_1 demande le sémaphore pris par P_2, la priorité de P_2 est augmentée au niveau de celle de P_1 (si $prio_1 > prio_2$).

Ordonnancement

Ordonnacement de tâches apériodiques

Pour des systèmes temps réel, il n'est pas raisonnable de se restreindre à l'usage de tâches périodiques. Pour gérer des événements, on doit utiliser des processus apériodiques. La difficulté consiste à faire cohabiter ces deux types de tâches sans remettre en cause les acquis de la politique d'ordonnancement.
Solutions :

- Affecter la priorité la plus basse aux tâches apériodiques (pas forcément en adéquation avec le cahier des charges, échéances difficiles à assurer à coup sûr mais impact sur les tâches périodiques quasi-nul)

- Rajouter un processus périodique (appelé serveur) dont la fonction est de s'occuper des tâches apériodiques.

Ordonnacement de tâches apériodiques

Serveur de scrutation :

- La tâche-serveur a souvent une priorité élevée et on lui attribut une capacité C_{serv} en fonction des tâches apériodiques qu'elle doit gérer.
- 2 parties : scruter à intervalles de temps régulier T_{serv} la présence de tâches apériodiques prêtes, et prendre la main si la file d'attente est non-vide.
- Quand une tâche apériodique demande l'UC, elle sera servie au mieux lors de la période de scrutation suivante.
- Il est possible d'utiliser plusieurs serveurs de scrutation en parallèle.

Ordonnancement

Ordonnacement de tâches apériodiques

Serveur de scrutation :

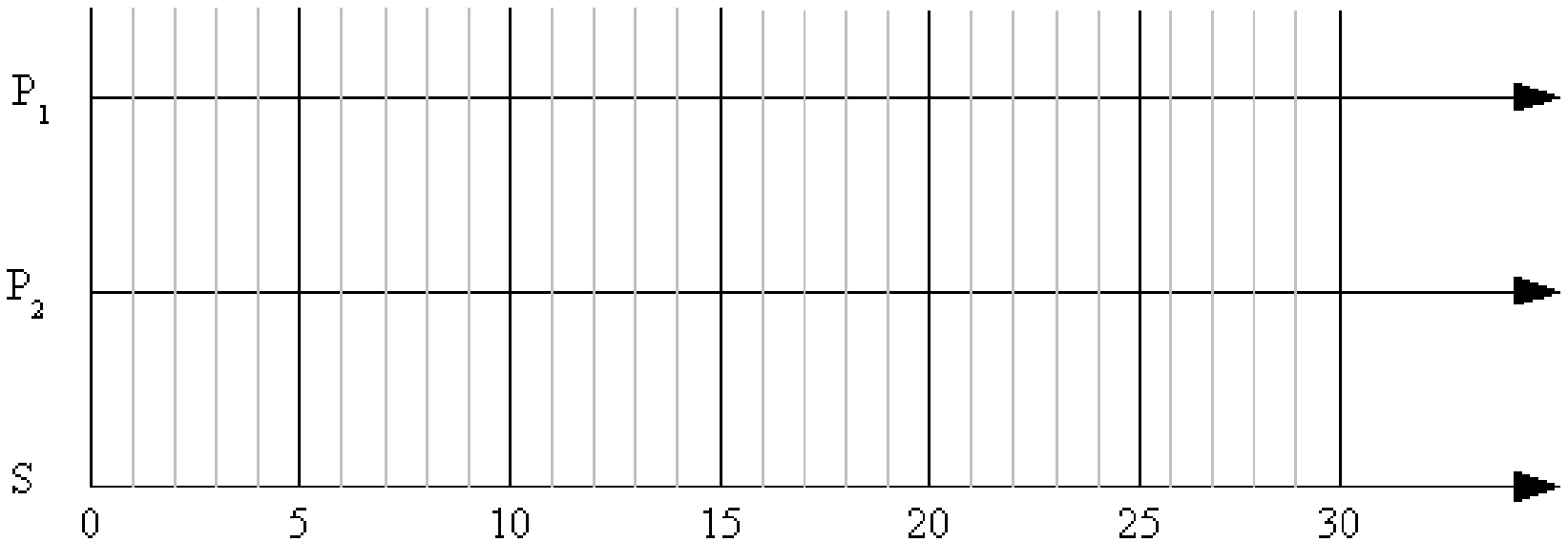
- Condition suffisante de RMA avec un serveur : $\sum_{i=1}^{n} \frac{C_i}{T_i} + \frac{C_{serv}}{T_{serv}} \leq (n+1)(2^{\frac{1}{n+1}} - 1)$.
- $+$: simple à mettre en oeuvre et test d'ordonnançabilité facilement adaptables.
- - : le serveur relâche sa capacité C_{serv} si aucun événement déclencheur de tâche apériodique n'est arrivé durant la période de scrutation (temps de réponse aux tâches apériodiques pas très bon).

Ordonnancement

Ordonnacement de tâches apériodiques

Serveur de scrutation : Exemple. Soient deux tâches P_1 et P_2 et un serveur S ordonnancés par RMA. On a $C_1 = 4$, $T_1 = 20$, $C_2 = 12$, $T_2 = 30$, $C_{serv} = 4$ et $T_{serv} = 10$. Soient deux événements e_1 et e_2 arrivant aux instants $t = 4$ et $t = 12$ et déclanchant deux tâches apériodiques de capacités 3 et 4 respectivement.

- Remplir de chronogramme.

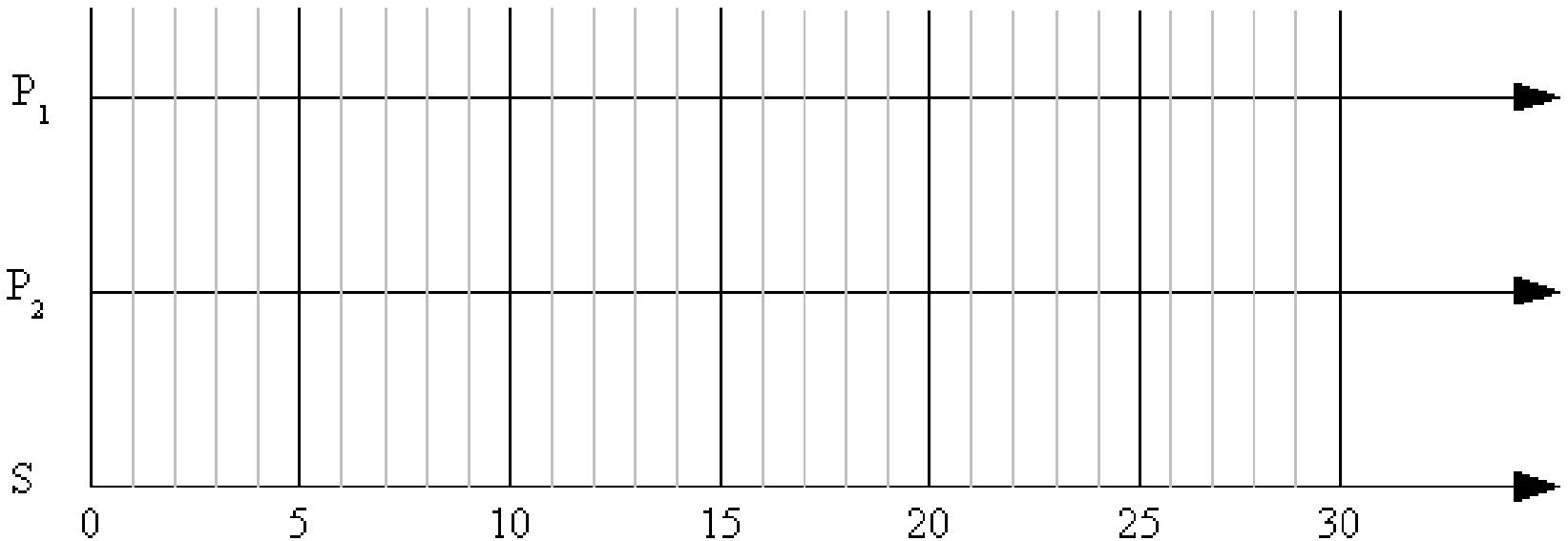


Ordonnancement

Ordonnacement de tâches apériodiques

Serveur de scrutation : Exemple.

- Remplir de chronogramme en considérant que les tâches apériodiques ne sont plus prises en compte par le serveur S mais simplement exécutées quand l'UC n'est pas prise par une tâche périodique. Conclure.



Ordonnancement

Ordonnacement de tâches apériodiques

Serveur différé :

- Même principe que le serveur de scrutation mais la capacité C_{serv} n'est pas directement abandonnée si aucune tâche apériodique n'est prête en début de période T_{serv}.
- Il abandonne sa priorité aux tâches périodiques tant qu'aucune tâche apériodique n'est prête. Si une tâche apériodique devient prête en cours de période de scrutation, alors le serveur reprend la main.

Ordonnancement

Ordonnacement de tâches apériodiques

Serveur différé :

- Condition suffisante de RMA avec un serveur différé : $\sum_{i=1}^{n} \frac{C_i}{T_i} \leq (n+1)(\left(\frac{U_{serv}+2}{U_{serv}+1}\right)^{\frac{1}{n+1}} - 1)$ si $T_{serv} \leq \min_i T_i$, $2T_{serv} \geq \max_i T_i$ et $T_{serv} + C_{serv} \leq \max_i T_i$.
- $+$: temps de réponse aux tâches apériodiques meilleur que pour le serveur de scrutation.
- - : plus complexe à mettre en place, phénomène d'exécution "dos à dos".

Ordonnancement

Ordonnacement de tâches apériodiques

Serveur différé : Exemple. Soient deux tâches P_1 et P_2 et un serveur différé S ordonnancés par RMA. On a $C_1 = 4$, $T_1 = 12$, $C_2 = 6$, $T_2 = 16$, $C_{serv} = 2$ et $T_{serv} = 10$. Soient les événements e_1, e_2 et e_3 arrivant aux instants $t = 2$, $t = 14$ et $t = 28$ et déclanchant des tâches apériodiques de capacités 1, 4 et 2 respectivement.

- Remplir de chronogramme (représenter la capacité $C_{serv}(t)$).

Ordonnancement

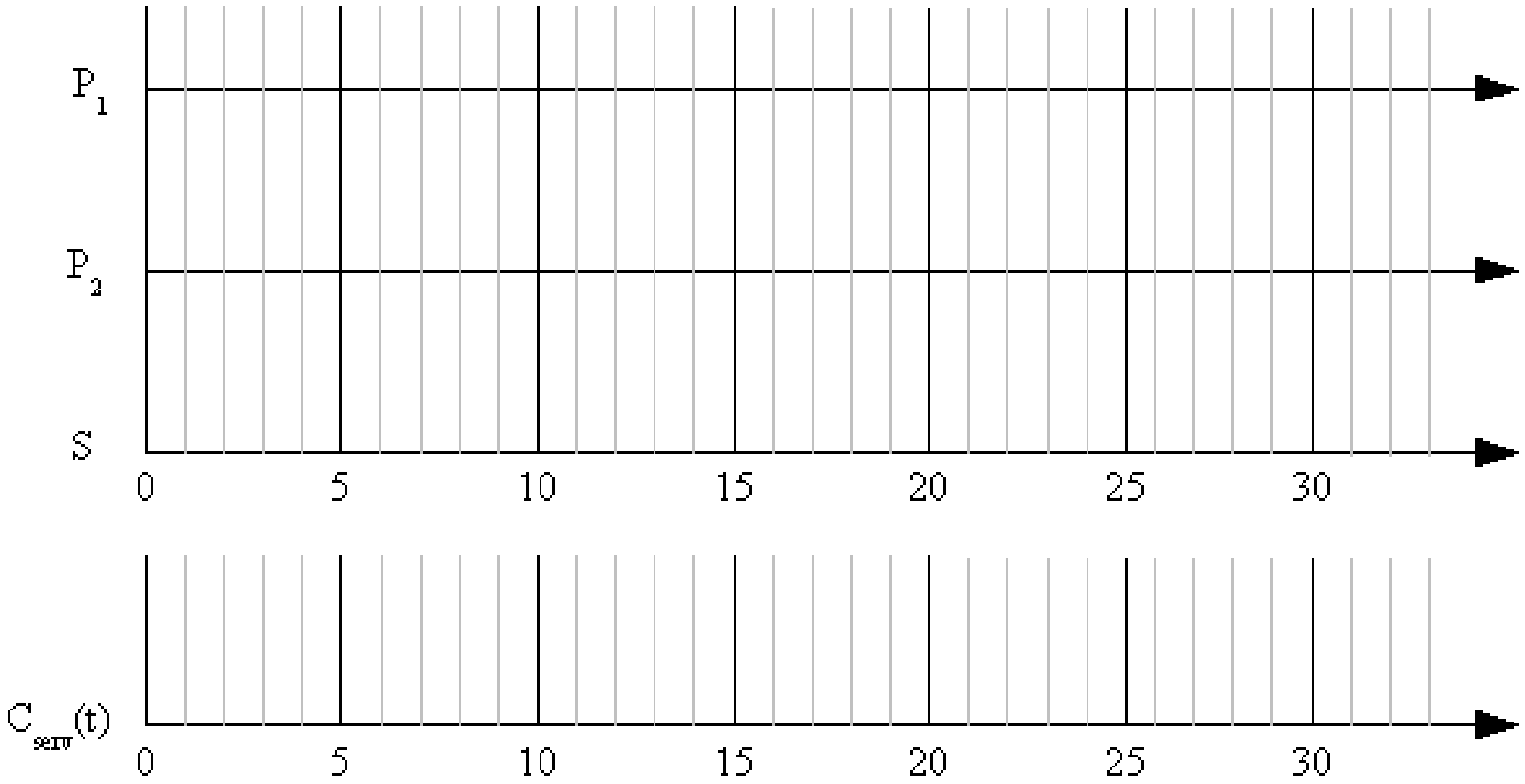
Ordonnacement de tâches apériodiques

Serveur différé : Exemple d'éxecution "dos à dos". Soient la tâche P_1 et le serveur différé S ordonnancés par RMA. On a $C_1 = 2$, $T_1 = 7$, $C_{serv} = 3$ et $T_{serv} = 6$. Soit l'événement e_1 arrivant à l'instant $t = 21$ et déclenchant une tâche apériodique de capacité 6.

- Remplir de chronogramme (représenter la capacité $C_{serv}(t)$).

Ordonnancement

Ordonnacement de tâches apériodiques



Ordonnancement

Ordonnacement de tâches apériodiques

Serveur sporadique :

- Même principe que le serveur différé mais
  - sa capacité n'est pas réallouée à période fixe, mais après un délai fixe Δ suivant l'instant où le serveur a pris la main. (Δ est souvent appelée période de réapprovisionnement).
    instant de réapprovisionnement = Δ+ instant de démarrage du serveur.
  - la capacité renouvelée est égale à la capacité consommée.

Ordonnancement

Ordonnacement de tâches apériodiques

Serveur sporadique :

- Condition suffisante de RMA avec un serveur sporadique : $\sum_{i=1}^{n} \frac{C_i}{T_i} + \frac{C_{serv}}{\Delta} \leq (n+1)(2^{\frac{1}{n+1}} - 1)$ (cas le pire : serveur utilise toute sa capacité autant que possible).
- $+$: temps de réponse aux tâches apériodiques possiblement meilleur que pour le serveur différé, pas d'éxection "dos à dos".
- - : plus complexe à mettre en place.

Ordonnancement

Ordonnacement de tâches apériodiques

Serveur sporadique : Exemple. Même énoncé que pour le serveur différé mais avec un serveur sporadique tel que $\Delta = 6$.