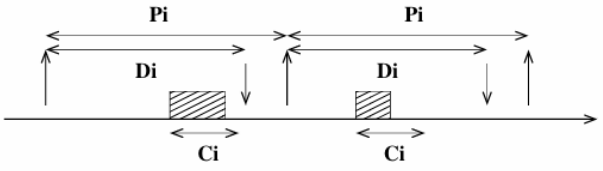
Ordonnancement temps réel

Définitions

A Worst-Case Execution Time (WCET) : Ci → Safe upper bound on the execution time of one job of the task

A release period : Pi → (Minimum) duration between consecutive job releases of the task

A relative deadline : Di → Maximum allowed response time for each job of the task



Static Priority

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

Rate Monotonic

- The smaller the period of a task, the higher its priority.

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

Deadlines Monotonic

- The smaller the deadline of a task, the higher its priority

Earliest Deadline First

- If The rate or deadlines monotonic don’t work, we use EDF

$$U = \sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1$$

A-Periodic Jobs

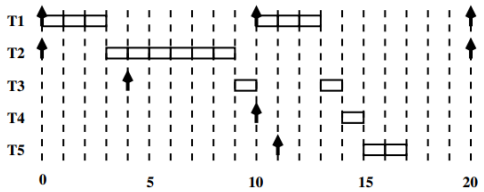
Background

- Periodic tasks are scheduled using Rate or Deadline Monotonic or EDF.
- A-Periodic jobs are scheduled following a ”First Come First Served” policy (Quand il y’a de l’espace)

- Task features

	(first) occurence	WCET	Deadline	Period
T_1	0	3	10	10
T_2	0	6	20	20
T_3	4	2		
T_4	10	1		
T_5	11	2		

- Task scheduling, assuming rate monotonic



‘ResponseTime = TaskEndingTime - WakeUpTime *(réveil dans le tableau)**'***

T3 = 14 - 4 = 10

T4 = 15 - 10 = 5

T5 = 17 - 11 = 6

Serveur :

Caractérisé par :

- une période Ps (fixe pour le serveur)
- un temps d’exécution Cs (fix pour le serveur) (=budget)
- un temps de réveil r0 = 0
- le serveur est ordonnancé suivant le même algorithme que les tâches périodique

Serveur de scrutation (Polling)

- On attend l’activation du serveur après chaque période.
- Si le serveur s’active et il n’y a aucune tâche en attente, le serveur perd le processeur, et perd sa capacité : Si à un instant une tâche apériodique vient, il doit attendre la prochaine activation du serveur.

Serveur de ajournable (Deferrable)

- Comme scrutation mais cette fois-ci, la capacité du serveur n’est pas gaspillé si lors de l’activation du serveur il n’y a aucune tâche en attente.
- Si une tâche arrive plus tard après l’activation du serveur, et que la période n’est pas finie et que le budget (capacité) n’est pas épuisé, la tâche se déroule.

Serveur de sporadique

Comme ajournable mais :

NextActivationPeriod (Next Server Period) = TaskArrivingTime + Ps

Précédence

Rate Monotonic

- Activation dates and deadlines are modified offline for each task T_j

$$r_j^* = max(r_j, r_i^*) \quad \text{for every } T_i \rightarrow T_j$$
$$D_j^* = D_j - (r_j^* - r_j)$$

- Different priorites for tasks with the same period: T_i has a higher priority than T_j if $T_i \rightarrow T_j$

Earliest Deadline First

- Modification of the activation date of T_j

$$r_j^* = max(r_j, max(r_i^* + C_i)) \quad \text{for all } T_i \rightarrow T_j$$

- Modification of the deadline of T_j

$$d_j^* = min(d_j, min(d_k^* - C_k)) \quad \text{for all } T_j \rightarrow T_k$$

r*

- On commence par les tâches qui n'ont pas de prédécesseur.
→ Leur r* reste inchangé.
- Si une tâche a 2 prédécesseurs ou plus, on prend le rj* maximale.

d*

- On commence par les tâches qui n'ont pas de successeur.
→ Leur d* reste inchangé.

Partage de ressources

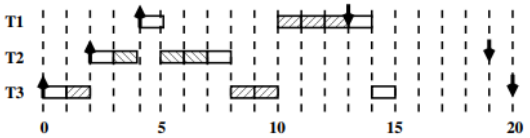
Ressource critique : pas de possibilité d'utilisation simultanée par plusieurs tâches à la fois.
→ On utilise toujours EDF.

Priority Inversion

Une tâche ne peut pas accéder à une ressource tant qu'elle est alloué par une autre tâche.

	First release	WCET	Deadline	Period
T_1	4	5	9	20
T_2	2	5	17	20
T_3	0	5	20	20

Resource utilisation		
R_1	R_1	R_1
R_2	R_2	R_2
R_1	R_1	R_1

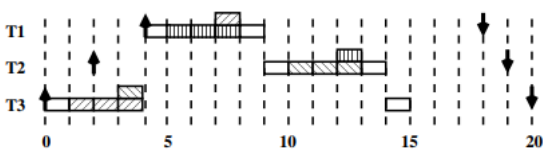


Super Priority

Une tâche qui alloue une ressource a toujours la priorité jusqu'à ce qu'il finisse l'allocation de la ressource.

	First release	WCET	Deadline	Period
T_1	4	5	14	20
T_2	2	5	17	20
T_3	0	5	20	20

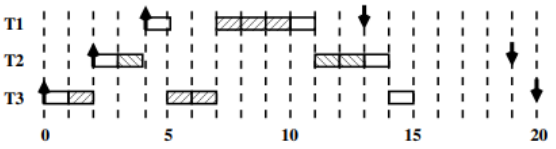
Resource utilisation		
R_3	R_3	R_1R_3
R_2	R_2	R_2R_3
R_1	R_1	R_1R_2



Priority Inheritance Protocol

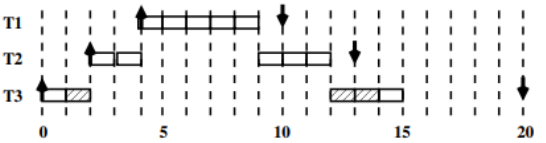
Si une tâche T1 prioritaire demande une ressource qui est déjà allouée par T2, la tâche non prioritaire T2 hérite la priorité jusqu'à ce qu'il désalloue la ressource, pour que T1 reprennent le relais.

	First release	WCET	Deadline	Period	Resource utilisation
T_1	4	5	9	20	R_1 R_1 R_1
T_2	2	5	17	20	R_2 R_2 R_2
T_3	0	5	20	20	R_1 R_1 R_1

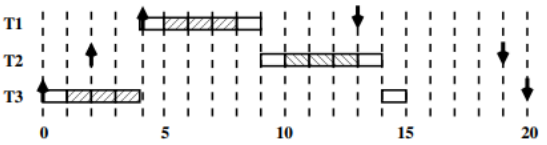


Stack-based protocol

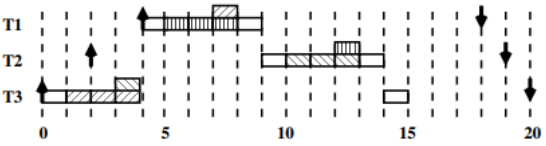
	First release	WCET	Deadline	Period	Resource utilisation
T_1	4	5	9	20	
T_2	2	5	17	20	
T_3	0	5	20	20	R_1 R_1 R_1



	First release	WCET	Deadline	Period	Resource utilisation
T_1	4	5	9	20	R_1 R_1 R_1
T_2	2	5	17	20	R_2 R_2 R_2
T_3	0	5	20	20	R_1 R_1 R_1



	First release	WCET	Deadline	Period	Resource utilisation
T_1	4	5	14	20	R_3 R_3 R_1R_3
T_2	2	5	17	20	R_2 R_2 R_2R_3
T_3	0	5	20	20	R_1 R_1 R_1R_2



→ On liste dans un tableau les priorités de chaque tâche par EDF.
→ On liste dans un autre tableau chaque ressource et son highest ceiling priority càd la priorité max des tâches qu'il l'alloue.

	r_0	WCET	D	P
T_1	5	3 : [ ] [R_1] [ ]	5	25
T_2	4	3 : [ ] [R_3] [ ]	8	25
T_3	3	6 : [R_4] [R_4] [R_4] [R_2R_4] [R_2R_4] [ ]	15	25
T_4	2	6 : [R_3] [R_3] [R_3] [R_3R_4] [R_3R_4] [ ]	22	25
T_5	0	6 : [ ] [R_1] [R_1R_2] [R_1R_2] [$R_1R_2R_3$] [ ]	25	25

Task	Priority
T1	5
T2	4
T3	3
T4	2
T5	1

Resource	Highest ceiling priority
R1	5
R2	3
R3	4
R4	3

Vérifier si une configuration est ordonnançable sur 2 processeur :

On calcule le pourcentage de monopolisation du processeur '***U=P/C' ,*** pour chaque tâche. Si on a 4 tâches par exemple, il faut trouver une combinaison entre les tâches pour que la somme de leur U ne dépasse pas 1, sinon la configuration n'est pas ordonnançable.

Systèmes opératoires Temps Réel

1. Dans le système temps réel OSEK, tous les objets sont définis statiquement dans un fichier OIL.
(a) L'ordonnanceur permet-il un comportement de type RM ? de type EDF ?
→ **Réponse :** Dans OSEK, les caractéristiques des tâches sont définies de manière statique dans le fichier OIL, ce qui permet à l'ordonnanceur de suivre un comportement de type RM (Rate Monotonic) grâce à des priorités fixes.
(b) L'ordonnanceur permet-il la préemption de tâche ?
→ **Réponse :** Oui, dans OSEK, la préemption des tâches est configurable : le mot-clé *'SCHEDULE = FULL / NONE'* détermine si une tâche peut être préemptée ou non.

2. Dans la norme ***ARINC 653***, la notion de partition est centrale.
(a) Comment l'exécution de ces partitions s'effectue-t-elle sur le calculateur avionique ?
→ **Réponse :** Le processeur alloue les partitions de façon déterministe et cyclique. L'OS définit un cycle de base appelé MAF (Major Time Frame) d'une durée fixe, au sein duquel chaque partition se voit attribuer une ou plusieurs tranches de temps, selon un fonctionnement similaire à l'algorithme Round-Robin (Tourniquet).
(b) Que se passe-t-il si deux processus d'une même partition souhaitent utiliser une même variable v ?
→ **Réponse :** Si l'accès à la variable vv est protégé par des sémaphores, un processus peut passer à l'état *'WAITING'* jusqu'à ce que le processus détenant la ressource termine son exécution.
(c) Même question en considérant deux processus de deux partitions différentes (un processus de chaque partition manipule les données d'une même variable v) ?
→ **Réponse :** Des communications sont possibles entre deux partitions différentes.

3. Dans la norme ***ARINC 653***, sont définis deux types de ségrégation : la ségrégation spatiale et la ségrégation temporelle.
(a) À quoi correspond la ségrégation spatiale ?
→ **Réponse :** La ségrégation spatiale garantit l'isolation mémoire entre les partitions pour empêcher qu'elles n'accèdent aux zones mémoire des autres.
(b) Comment est réalisée la ségrégation temporelle ?
→ **Réponse :** Elle est réalisée en attribuant des tranches de temps fixes à chaque partition dans un cycle déterministe défini par le MAF (Major Time Frame).
(c) Que garantissent ces deux ségrégations ?
→ **Réponse :** Elles garantissent l'isolation fonctionnelle et temporelle, assurant fiabilité, sécurité et absence d'interférences entre partitions.
(d) Pensez-vous que Linux (ou ses évolutions) propose ces ségrégations ? Si oui, quel(s) mécanisme(s) de Linux permettent de les réaliser ?
→ **Réponse :** Oui, Linux, avec des évolutions comme Xenomai ou PREEMPT-RT, utilise des cgroups pour la ségrégation spatiale et des politiques d'ordonnancement temps réel pour la ségrégation temporelle.

4. Le système ***Xtratum*** est une implantation de la norme ARINC 653 pour le domaine satellite.
(a) Qu'est-ce que le ***TSP*** et en quoi consiste-t-il ?
→ **Réponse :** Le TSP (Time and Space Partitioning) est un mécanisme garantissant l'isolation spatiale (mémoire) et temporelle (temps d'exécution) entre partitions, afin de prévenir les interférences entre applications critiques.
(b) Pourquoi ce type de solution est-elle adaptée pour les applications critiques telles que celles exécutées dans le domaine avionique ou satellite ?
→ **Réponse :** Le TSP assure la sécurité et la fiabilité en isolant les partitions, empêchant ainsi qu'une défaillance dans une application n'impacte les autres. Ceci est crucial pour les environnements avioniques et satellites où des erreurs peuvent être fatales.
(c) Le domaine automobile utilise OSEK. Une solution à base de TSP aurait-elle un intérêt ?
→ **Réponse :** Oui, une solution basée sur le TSP aurait un intérêt dans le domaine automobile pour isoler les fonctions critiques (ex. freinage, direction) et garantir leur exécution, même en cas de défaillance d'autres applications moins critiques

5. Quel est le rôle de l'instruction TerminateTask() ?
→ **Réponse :** Cette instruction permet la terminaison de la tache, celle-ci passe de l'état '*RUNNING'* à l'état '*READY'*.

6. Le problème d'inversion de priorité peut-il survenir en OSEK ?
→ **Réponse :** Dans OSEK, l'utilisation de sémaphores et le principe d'exclusion mutuelle impliquent qu'une tâche de priorité élevée peut être bloquée en attendant l'accès à une ressource détenue par une tâche de priorité inférieure. Cela peut entraîner un phénomène d'inversion de priorité.

7. A quoi servent les routines 'hook'?
→ **Réponse :** Les routines *hook* dans OSEK permettent d'exécuter des fonctions spécifiques à des événements prédéfinis, comme le démarrage ou l'arrêt d'une tâche, le passage en état d'erreur ou, dans ce cas, la gestion des interruptions pour incrémenter le compteur de *ticks*. Elles servent principalement à personnaliser le comportement du système sans modifier son noyau.

Langage OIL

1. Écrire, en langage OIL, la configuration d'un compteur nommé SysTimerCnt qui permet l'incrémentation d'un tick toutes les 3 ms, la période minimale d'une tâche étant d'un tick .

```
COUNTER SysTimerCnt {
    TYPE = HARDWARE;
    TICKSPERBASE = 3;  // Compteur qui génère un tick toutes les 3 ms
}
```

2. Écrire, en langage OIL, la configuration d'une alarme périodique, Alarm1, associée au compteur SysTimerCnt et qui réveille la tâche TASK1. La période de cette alarme est 5 ticks et sa 1 ère occurrence est produite à 1 tick.

```
ALARM Alarm1 {
    COUNTER = SysTimerCnt;          /* Associer au compteur */
    ACTION = ACTIVATETASK {         /* Activer la tâche T1 */
        TASK = TASK1;
    };
    AUTOSTART = TRUE {              /* Démarrer automatiquement */
        ALARMTIME = 1;              /* Délai initial de démarrage : r0 = 4 ms */
        CYCLETIME = 5;             /* Période : P = 15 ms */
    };
};
```

3. Quelle est la période, en milliseconde, de l'alarme Alarm1 ?
→ **Réponse : *P = TICKSPERBASE x CYCLETIME*** = *3 x 5 = 15 ms*

4. TASK2 a une période de 60ms. Quelle est la tâche prioritaire ?
→ **Réponse :** En RM celle qui a la période la plus courte donc TASK1.

5- Quel problème peut survenir lors de l'utilisation de la variable partagée obstacle dans le code ? Que faut-il faire, en OSEK, pour y remédier ?
→ **Réponse :** On peut préempter une tache au mauvais moment et mener à un état incohérent du système. Pour y remédier, on définit une ressource qui va protéger les accès en lecture et écriture à la variable protégée obstacle.

6.Ecrire, en langage OIL, la configuration des tˆaches TASK1 en considérant qu'elles possède la priorités la plus faible du système.

```
TASK TASK1 {
    PRIORITY = 1;               // Priorité la plus faible
    AUTOSTART = FALSE;          // La tâche ne démarre pas automatiquement
    ACTIVATION = 1;             // Une seule activation simultanée
    SCHEDULE = FULL;            // Tâche préemptive
    RESOURCE = SharedResource;  // Ressource utilisée par TASK1 et TASK2
}
```

7. Pour une tâche T1 qui se réveille à *r0 = 4* et une période *P = 15*, et en utilisant un timer qui s'incrémente toutes les millisecondes (TICKSPERBASE = 1), on a l'alarme A1 :

```
ALARM A1 {
    COUNTER = SysTimerCnt;      // Associer l'alarme au compteur SysTimerCnt
    ACTION = ACTIVATETASK {     // L'alarme active la tâche T1
        TASK = T1;
    };
    AUTOSTART = TRUE {          // L'alarme démarre automatiquement
        ALARMTIME = 4;          // Première activation après 4 ticks (4 ms)
        CYCLETIME = 15;         // Répétition toutes les 15 ticks (15 ms)
    };
}
```

8. Quelle(s) fonction(s) doit-on exécuter dans le code de T1 pour accéder à la ressource R1?

→ **Réponse :** Fonction à utiliser pour accéder à la ressource R1 par T1 est : GetResource(R1);

9. T1 : *r0 = 4*, *P=15*, et accède au ressource *R1*

```
COUNTER SysTimerCnt {
    MINCYCLE = 1;                   /* Incrémentation minimale (1 ms) */
    TICKSPERBASE = 1;               /* Une unité = 1 ms */
};
ALARM A1 {
    COUNTER = SysTimerCnt;          /* Associer au compteur */
    ACTION = ACTIVATETASK {         /* Activer la tâche T1 */
        TASK = T1;
    };
    AUTOSTART = TRUE {              /* Démarrer automatiquement */
        ALARMTIME = 4;              /* Délai initial de démarrage : r0 = 4 ms */
        CYCLETIME = 15;             /* Période : P = 15 ms */
    };
};
TASK T1 {
    PRIORITY = 1;                   /* Priorité la plus faible */
    ACTIVATION = 1;                 /* Une seule activation à la fois */
    SCHEDULE = FULL;                /* Tâche préemptive */
    AUTOSTART = FALSE;              /* La tâche ne démarre pas automatiquement */
    RESOURCE = R1;                  /* Accès à la ressource R1 */
};
```