Une approche synchrone à la conception de systèmes embarqués temps réel

Dumitru Potop-Butucaru

dumitru.potop@inria.fr

cours EIDD, 2023, 4ème séance

Contenu de ce cours

- Ordonnancement en ligne préemptif
  - Théorie
  - Programmer un ordonnanceur
    - En Heptagon
- Préparation du TP
  - Programmation de Rate Monotonic (RM) et Earliest Deadline First (EDF)

Jargon temps réel

- **Tâches** = Fonctions calculant les réactions (ou juste des bouts de ces réactions)
- Moments importants dans la vie d'une tâche :
  - **Arrivée (arrival)** = date où l'exécution d'une tâche **peut** commencer
  - **Échéance (deadline)** = date après l'arrivée où la tâche doit être finie.
  - **Démarrage (start)** = date après l'arrivée et avant l'échéance où la tâche commence à l'exécuter. Peut ne pas exister en cas d'échéance manquée.
  - **Fin (end)** = date après le démarrage et avant l'échéance où les calculs se terminent. Peut ne pas exister en cas de d'échéance manquée.
  - **Interruption** – suspensive (^Z) ou définitive (^C)
  - **Reprise (resume)** – avec préservation de l'état (contexte)
- Types de tâches:
  - **Périodiques** – arrivent à des intervalles fixes
  - **Sporadiques** – arrivent avec une distance minimale entre elles
  - Apériodiques – peuvent arriver sans restrictions

Ordonnancement temps-réel

- Allocation des ressources aux tâches dans le but d'assurer le respect des échéances
  - Démarrage/interruption/reprise de tâches
  - Mais aussi allocation mémoire, allocation de ressources I/O...
- On peut le faire :
  - En ligne - lors de l'arrivée des stimuli, suivant des politiques d'ordonnancement généralistes
  - Hors ligne - dates de départ choisies avant exécution, politique d'ordonnancement spécifique au système

…mais la limite entre les 2 n'est pas exactement définie:

- Une politique « en ligne » a des paramètres que l'on peut varier avant l'exécution
- Une politique « hors ligne » peut ne pas couvrir des aspects comme l'allocation des bancs mémoire ou même le choix du processeur, qui sont réalisées alors en ligne.

Encore une fois, c'est l'état d'esprit qui compte.

Ordonnancement en ligne

- Implantation événementielle
  - Plusieurs signaux/interruptions peuvent déclencher des calculs. Problèmes de synchronisation
- Algorithmes classiques
  - RM (rate monotonic), FP (fixed priority), EDF (earliest deadline first), DM (deadline monotonic), etc.
- Avantages:
  - Réactions très rapides à des événements prioritaires.
  - Robustesse aux variations temporelles (temps d'exécution, dates d'arrivée des tâches).
- Problèmes:
  - Nécessitent souvent des marges importantes avec les critères d'ordonnançabilité classiques (30% pour RM)
  - Non-déterminisme temporel, plus difficile à vérifier/simuler/tester
  - Exécution conditionnelle difficile à exploiter

Algorithmes d'ordonnancement en ligne

- A lire:
  - **Liu & Layland 1973**: Scheduling algorithms for multiprogramming in a hard real-time environment.
  - **G. Buttazzo 2005**: Hard real-time computing systems

- Modélisation classique:
  - n tâches $\tau_1,\ldots,\tau_n$. Chaque tâche τ_i a:
    - Une période T_i
      - Tâches périodiques (avec ou sans gigue et/ou dérive) ou sporadiques
    - Une durée (capacité) C_i
    - Une échéance d_i – calculée à partir de chaque arrivée
    - Une date de premier démarrage s_i^0
  - Exécution sur un processeur **séquentiel**
  - Comment ordonner l'exécution des tâches pour **garantir** le respect les échéances?
    - Qu'est-ce qui se passe si une échéance est manquée ?

Algorithmes d'ordonnancement en ligne

- Les algorithmes « équitables »
  - FIFO, Round Robin, Weighted Round Robin, Resource reservation/Real-Time Calculus
  - Pas de prise en compte de l'état de l'application (sauf changements de configuration coûteux)
  - Mais: modularité (pour la réservation de ressources)

- Ordonnancement préemptif à priorités
  - Chaque tâche τ_i a une priorité $prio_i$
  - A chaque instant, la tâche qui s'exécute est **une** des tâches actives de priorité maximale
    - Une tâche plus prioritaire interrompt les tâches moins prioritaires
  - Permet de donner plus de temps aux tâches qui en ont besoin, en fonction de l'état du système
    - État = activation des tâches

Ordonnancement préemptif

- Comment choisir les priorités
  - Statiquement (offline) = Priorité fixe (FP)
    - RM = Rate Monotonic: $T_i > T_j \Rightarrow prio_i < prio_j$
    - DM = Deadline Monotonic: $d_i > d_j \Rightarrow prio_i < prio_j$
    - …
  - Dynamiquement (online)
    - EDF = Earliest Deadline First (la tâche à échéance la plus proche est prioritaire)
    - LLF = Least Laxity First (la tâche avec le moins de « mou » = distance jusqu'à échéance – temps restant à exécuter sur la capacité)

    …
- Règles simples, faciles à implémenter, à l'aide d'interruptions
  - Support matériel fourni par tous les processeurs

Ordonnancement préemptif

- Est-ce que toutes les tâches vont respecter leurs écheances?
  - Condition nécessaire : chaque tâche, prise séparément, doit pouvoir s'exécuter :

$$C_i \leq T_i \quad \text{ou} \quad \frac{C_i}{T_i} \leq 1$$

    - Jamais une nouvelle instance n’arrive avant que l'ancienne soit finie
  - Notion centrale : **utilisation (charge) du processseur**
    - Exemple : La tâche *t* a une période de 50ms et une durée de 10ms. Alors, *t* utilise 10/50 = 20% de la capacité de calcul du processeur.

Ordonnancement préemptif

- Est-ce que l'ensemble des tâches vont respecter leurs échéances?
  - Notion centrale : **utilisation (charge) du processeur**

$$\sum_{i=1}^{n} \frac{C_i}{T_i}$$

    - On vise à garantir le respect des échéances => utilisation des durées au pire cas C_i
    - $\frac{C_i}{T_i}$ = pourcentage du temps CPU utilisé (au pire cas) par τ_i
  - Condition **nécessaire** sur un mono-processeur :

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

  - Condition **nécessaire** sur n processeurs identiques:

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

Ordonnancement préemptif

- Est-ce que toutes les tâches vont respecter leurs écheances?
  - **Critère d'ordonnançabilité = condition suffissante**
  - Liu & Layland 1973: **Si le coût des préemptions est 0, et $T_i = d_i$,** alors :
    - FP/RM/DM: Condition suffisante :

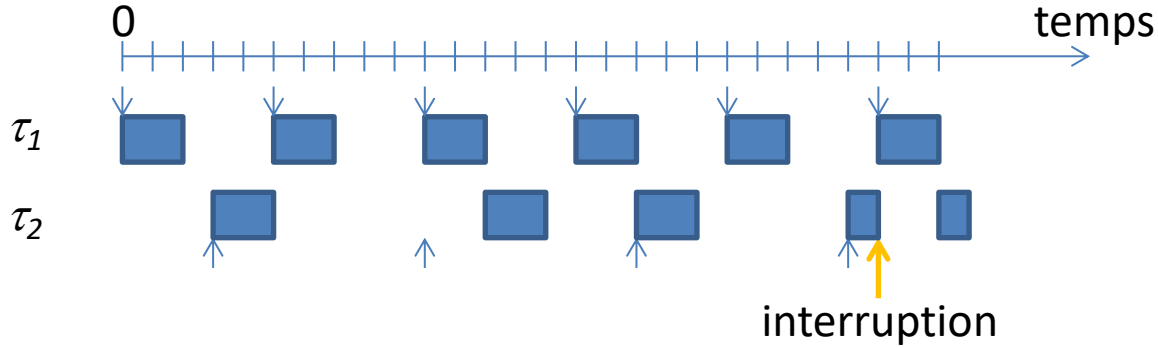
$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n \cdot (2^{1/n} - 1) \qquad \lim_{n \to \infty} \left(n \cdot (2^{1/n} - 1) \right) = \ln 2 \cong 0{,}69$$

    - EDF: Condition nécessaire et suffisante (optimal):

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

Ordonnancement préemptif

- Exemple: n=2, T_1 =5, C_1 =2, T_2 =7, C_2 =2
- RM: $prio_1$ =1, $prio_2$ =0

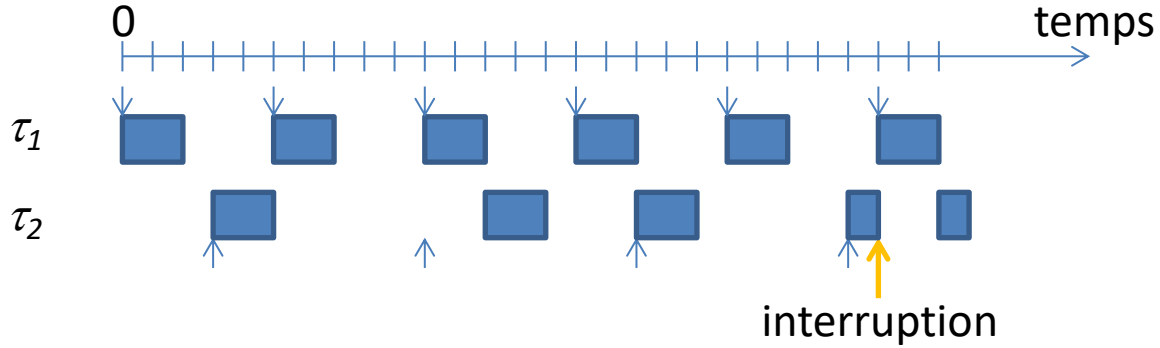


$$\sum_{i=1}^{n} \frac{C_i}{T_i} < n\left(2^{\frac{1}{n}} - 1\right) < 1$$

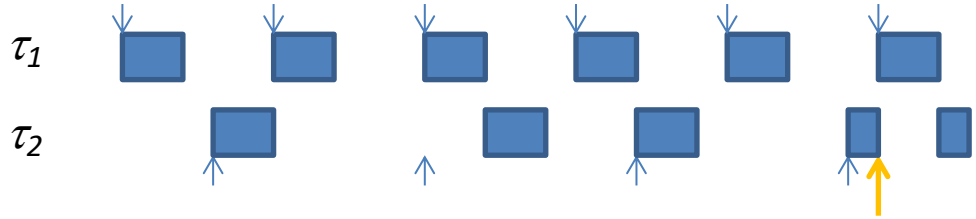
~0.68% CPU charge

Ordonnancement préemptif

- Exemple: n=2, T_1 =5, C_1 =2, T_2 =7, C_2 =2
- RM: $prio_1$ =1, $prio_2$ =0



- EDF:

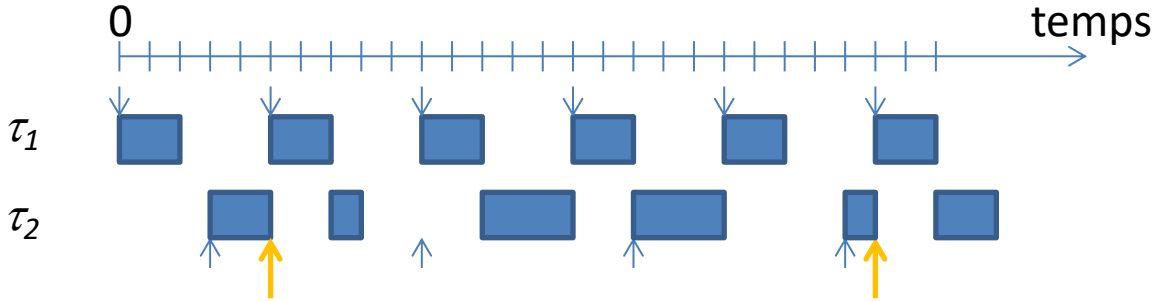


$$\sum_{i=1}^{n} \frac{C_i}{T_i} < n\left(2^{\frac{1}{n}} - 1\right) < 1$$

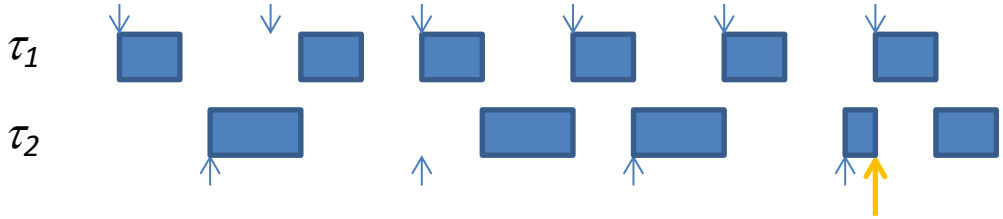
~0.68% CPU charge

Ordonnancement préemptif

- Exemple: n=2, T_1 =5, C_1 =2, T_2 =7, C_2 =3
- RM: $prio_1$ =1, $prio_2$ =0



- EDF:

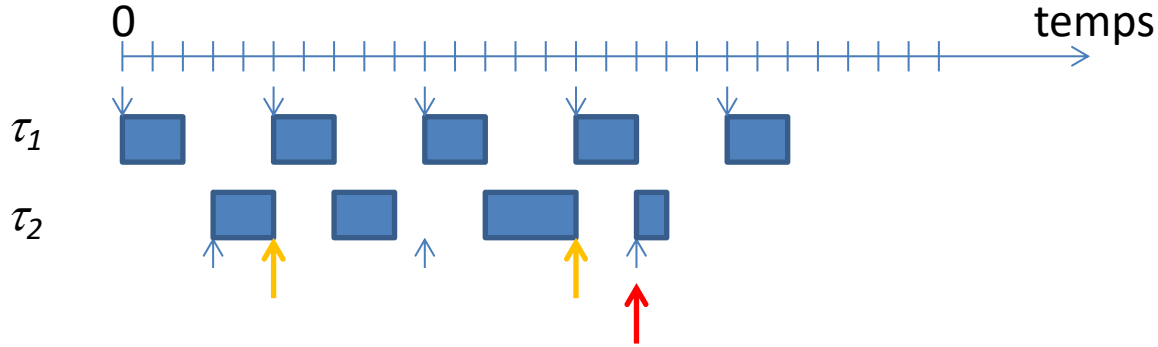


$$\sum_{i=1}^{n} \frac{C_i}{T_i} < n\left(2^{\frac{1}{n}} - 1\right) < 1$$

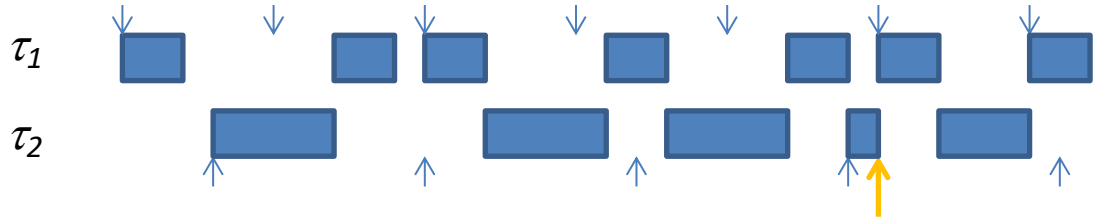
~0.82% CPU charge!

Ordonnancement préemptif

- Exemple: n=2, T_1 =5, C_1 =2, T_2 =7, C_2 =4
- RM: $prio_2$ =0, $prio_1$ =1



- EDF:



$$n \cdot (2^{1/n} - 1) < \sum_{i=1}^{n} \frac{C_i}{T_i} < 1$$

>0.97% CPU charge!

Ordonnancement préemptif

- Toutes les analyses d'ordonnançabilité font des hypothèses:
  - Durées des opérations:
    - au pire cas (WCET), parfois dans le meilleur des cas (BCET)
  - Coût 0 (ou fixe) pour les préemptions, communications, etc.
- Que se passe-t-il si ces hypothèses ne sont pas respectées ?
  - FP (RM/DM):
    - Les tâches plus prioritaires gardent l'accès aux ressources
      - bien pour les systèmes critiques, lien priorité-criticité
  - EDF:
    - On n'en sait rien (pas bien du tout)

Ordonnancement préemptif

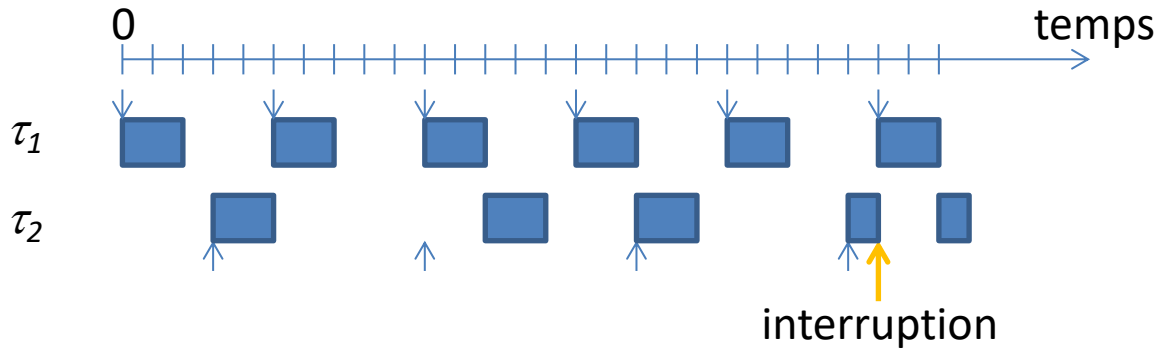
- Beaucoup de sources d'incertitude dans l'analyse
  - Date de premier départ des tâches
  - Durée effective de chaque tâche
    - Une tâche se terminant plus tôt n'améliore pas toujours les temps de réponse : notion d'**anomalie temporelle**
  - Et en général, c'est pire : gigue, allocation à choisir sur un multiprocesseur, allocation mémoire…
- Incertitude+analyse au pire cas+complexité => coût en précision de l'analyse

Contenu de ce cours

- Ordonnancement en ligne préemptif
  - Théorie
  - Programmer un ordonnanceur
    - En Heptagon
- Préparation du TP
  - Programmation de Rate Monotonic (RM) et Earliest Deadline First (EDF)

Ordonnanceur

- Exemple: n=2



  - T_1 =5, C_1 =2, d_1=T_1, s_0^1=0
  - T_2 =7, C_2 =2, d_2=T_2, s_0^2=3

- Politique RM: $prio_1$ =1, $prio_2$ =0

Ordonnanceur

- Modélisation Heptagon des tâches

```
type task_attributes =
     { period      : int ;
       capacity    : int ;
       first_start : int }
```

Ordonnanceur

- Modélisation Heptagon des tâches

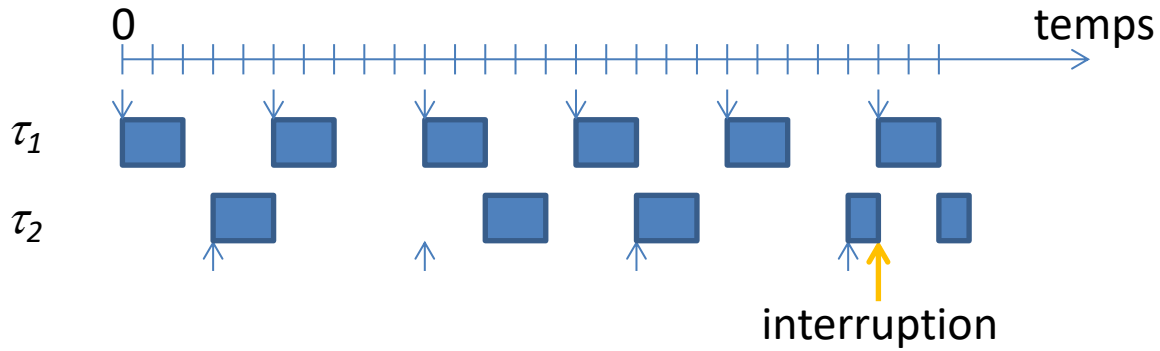
```
type task_attributes =
     { period      : int ;
       capacity    : int ;
       deadline.   : int ;
       first_start : int }
```

  - $T_1 = 5$, $C_1 = 2$, $d_1 = T_1$, $s_0^1 = 0$
  - $T_2 = 7$, $C_2 = 2$, $d_2 = T_2$, $s_0^2 = 3$

```
const ntasks : int = 2
const tasks : task_attributes^ntasks =
  [{ period=5; capacity=2; deadline=5; first_start=0 },
   { period=7; capacity=2; deadline=7; first_start=3 }]
```

Ordonnanceur

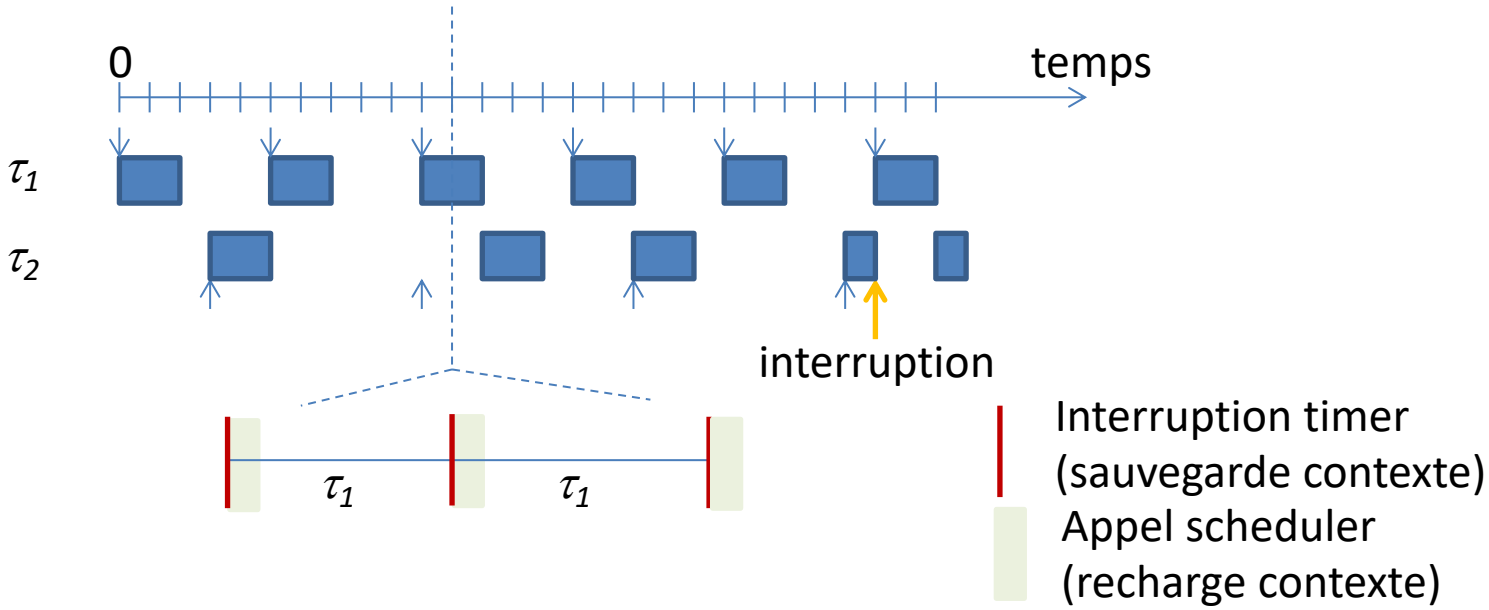
- Etat d'ordonnancement



  - *Date courante*
  - *Tâches en cours d'exécution*
- Reste constant entre appels à l'ordonnanceur

Ordonnanceur

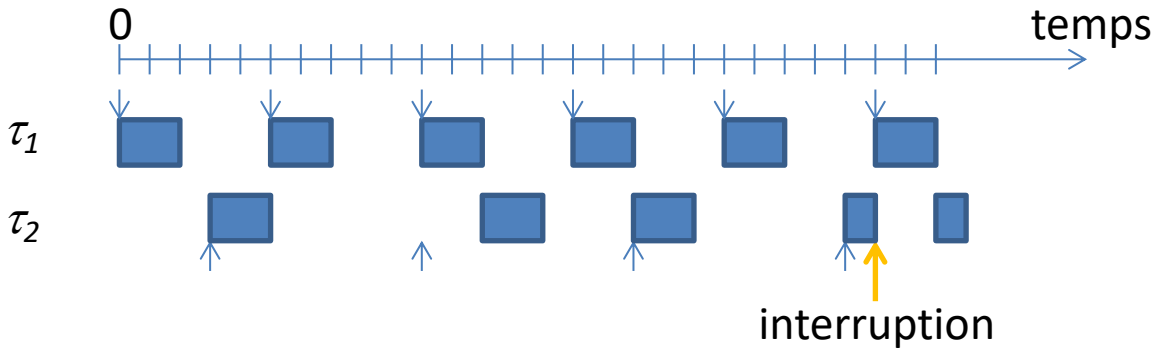
- Etat d'ordonnancement



- Reste constant entre appels à l'ordonnanceur

Ordonnanceur

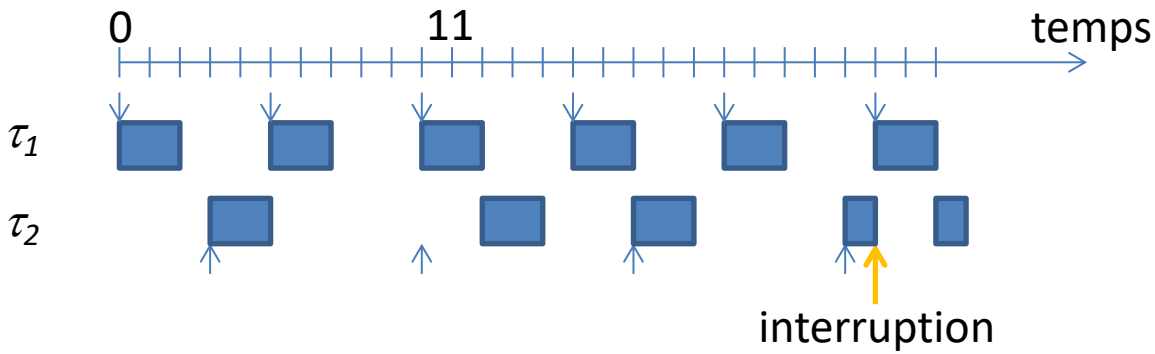
- Etat d'ordonnancement




```
type task_state = Running | Ready | Waiting
type task_status =
     { status           : task_state ;
       current_deadline : int ;
       left             : int }
type scheduler_state =
     { current_date : int ;
       tasks        : task_status^task_number }
```

Ordonnanceur

- Etat d'ordonnancement

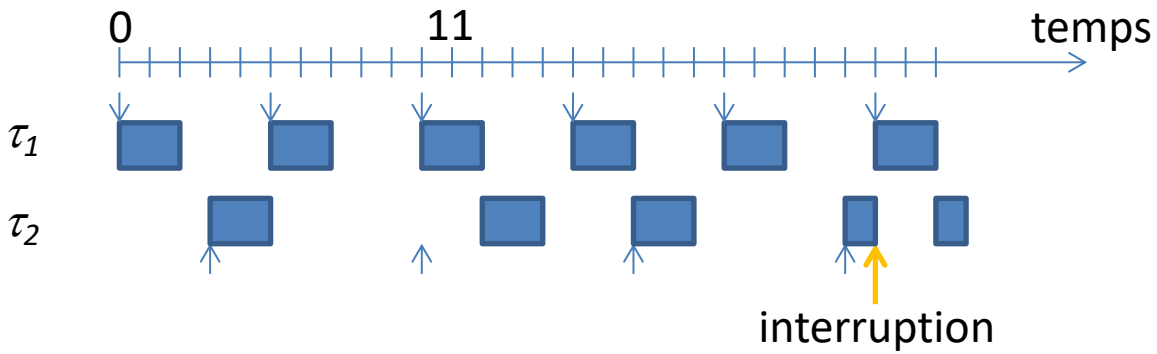


A la date 11, *après* l'appel au scheduler:

```
{ current_date = 11;
  tasks =
    [ {status=Running; current_deadline=15; left=1},
      {status=Ready  ; current_deadline=17; left=2}] }
```

Ordonnanceur

- Etat d'ordonnancement

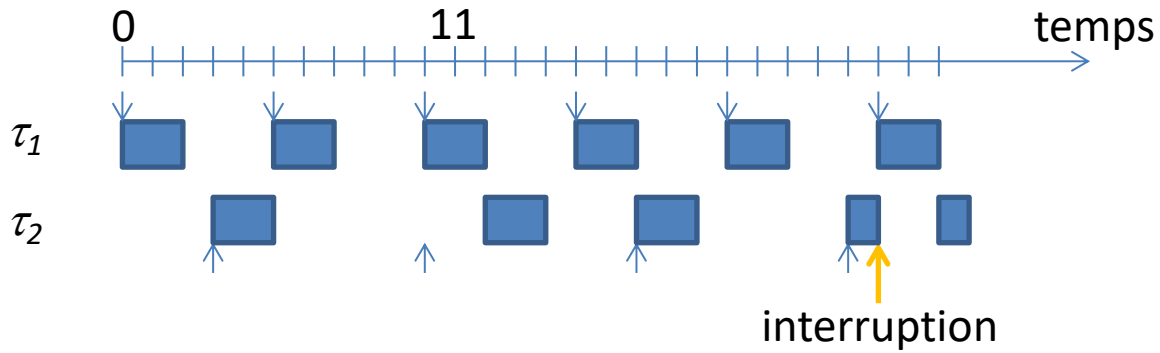


Etat initial de l'ordonnancement, avant le premier appel :

```
const init_sstate : scheduler_state =
 { current_date = -1 ;
   tasks =
      {status=Waiting;current_deadline=0;left=0}^2 }
```

Ordonnanceur

- Ordonnanceur déclenché périodiquement (à chaque pas de temps)




```
fun scheduler(si:scheduler_state) returns (so:scheduler_state)
```

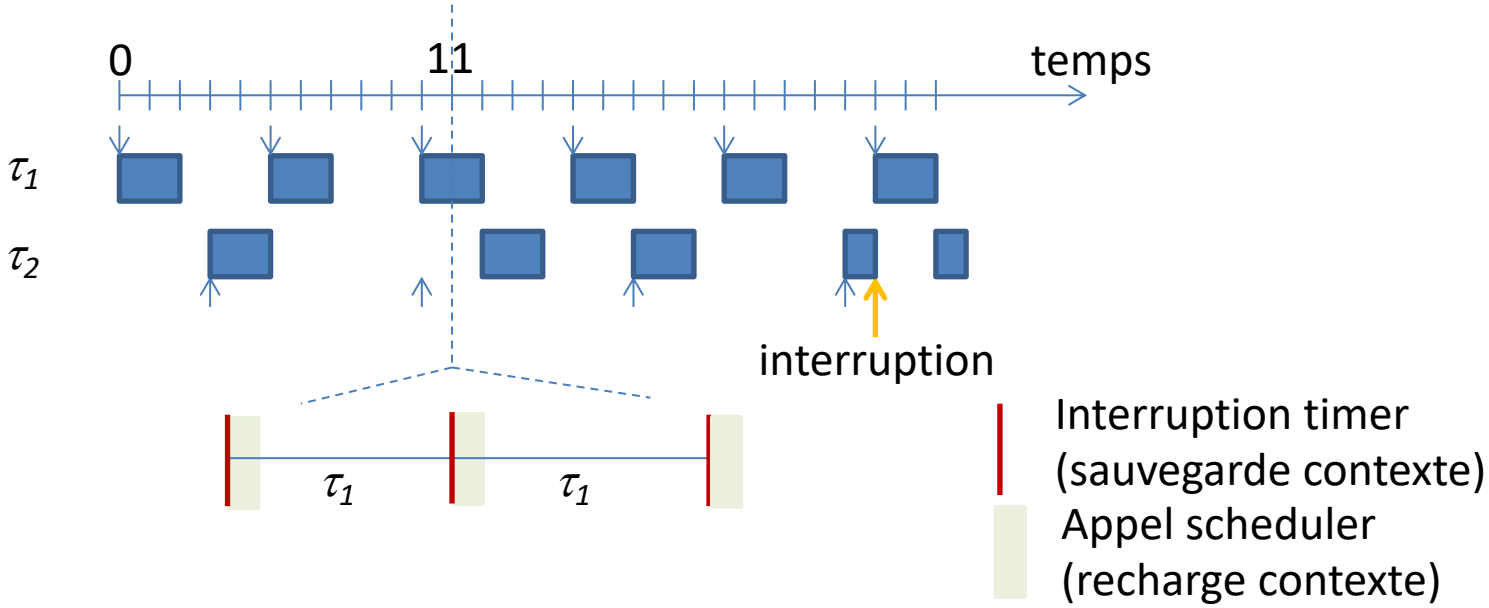
État précédent

État pour l'intervalle de temps suivant

```
node main() returns () var state: scheduler_state ;
let
  state = init_state fby scheduler(state);
tel
```

Ordonnanceur

- Ordonnanceur



Ordonnanceur

- Actions réalisées par l'ordonnanceur
  1. Simuler l'exécution du cycle précédent
     - Tâches qui se terminent: Running->Waiting
     - Ou qui ne se terminent pas: Running->Ready
       - Et « left » est décrémenté de 1 (modélisation)
  2. Déterminer s'il y a des échéances manquées
     - Error handling: message d'erreur, passage en état Waiting
  3. Déterminer s'il y a des arrivées de nouvelles instances à lancer
     - Simulation: choix d'une durée – utilisation de rand()
     - Passage en état Ready
  4. Politique d'ordonnancement: quelle tâche Ready devient Running ?

Ordonnanceur

- Actions réalisées par l'ordonnanceur

```
fun scheduler(si:scheduler_state) returns (so:scheduler_state)
var new_date : int ;
    tmp1,tmp2,tmp3,fin: task_status^ntasks ;
let
  new_date = si.current_date + 1 ; (* advance time by 1 *)
  tmp1 = map <<ntasks>> simulate (si.tasks) ;
  tmp2 = mapi<<ntasks>> check_deadline (new_date^ntasks,tmp1);
  tmp3 = map <<ntasks>> start_inst (new_date^ntasks,tmp2,tasks);
  fin  = rate_monotonic(tmp3) ; (* scheduling policy *)
  so   = { current_date = new_date; tasks = fin }
tel
```

Ordonnanceur

- Politique d'ordonnancement RM (1/2)

```
fun update_selected(ts:task_status;selected:int;tid:int)
                                        returns (tso:task_status)
let
  tso = if tid = selected then { ts with .status = Running }
        else ts
tel


fun rate_monotonic(ts:task_status^ntasks)
                            returns (tso:task_status^ntasks)
var selected : int ;
let
  selected = select_one_task(ts) ;
  tso = mapi<<ntasks>> update_selected (ts,selected^ntasks) ;
tel
```

Ordonnanceur

- Politique d'ordonnancement RM (2/2)

```
type select_acc = { tid : int; speriod : int }
fun select_aux(ts:task_status;ta:task_attributes;
               tid:int;acc:select_acc) returns (acc_o:select_acc)
let
  acc_o =
    if (ts.status = Ready) and (ta.period < acc.speriod) then
         { tid = tid; speriod = ta.period }
    else acc
tel
fun select_one_task(ts:task_status^ntasks) returns(selected:int)
var tmp : select_acc ;
let
  tmp = foldi<<ntasks>> select_aux
              (ts,tasks,{ tid = ntasks; speriod = int_max }) ;
  selected = tmp.tid ;
tel
```

Ordonnanceur

- Démarrage de nouvelles instances périodiques

```
fun start_inst(current_date:int;tsi:task_status;
                ta:task_attributes) returns (tso:task_status)
var c : bool ;
let
  c = (current_date-ta.first_start)%ta.period = 0 ;
  tso = merge c
          (true ->
            { status = Ready;
              current_deadline =
                  (current_date when c) + (ta.deadline when c);
              left = random(ta.capacity when c) })
          (false -> tsi whenot c)
tel
```

- random – c'est un simulateur, cela fixe les durées

Ordonnanceur

- Démarrage de nouvelles instances périodiques

```
fun start_inst(current_date:int;tsi:task_status;
                ta:task_attributes) returns (tso:task_status)
var c : bool ;
let
  c = (current_date-ta.first_start)%ta.period = 0 ;
  tso = merge c
          (true ->
            { status = Ready;
              current_deadline =
                  (current_date when c) + (ta.deadline when c);
              left =        ta.capacity when c  })
          (false -> tsi whenot c)
tel
```

- Traces obtenues sans « random »

Ordonnanceur

- Détection&gestion des échéances manquées

```
fun check_deadline(current_date:int;tsi:task_status;tid:int)
                                          returns (tso:task_status)
var c: bool ;
let
  c = (tsi.status = Ready)
        and (tsi.current_deadline = current_date) ;
  () = deadline_miss_log(current_date when c,tid when c) ;
  tso = if c then { tsi with .status = Waiting} else tsi ;
tel
```

- Miss => message (log) et arrêt de l'instance
  - Autres approches sont possibles

Ordonnanceur

- Simuler l'exécution

```
fun simulate(tsi:task_status) returns (o:task_status)
let
  o =
    if tsi.status = Running then
      if tsi.left <=1 then
        (* Normal termination, move to Waiting state *)
        { tsi with .status = Waiting }
      else
        (* No termination, yet *)
        { status = Ready;
          current_deadline = tsi.current_deadline ;
          left = tsi.left - 1 }
    else tsi
tel
```

- Dans un système réel, l'exécution décide des terminaisons

Ordonnanceur

- Mise en œuvre :
  - Structures de données : scheduler_data.ept
  - Déclarations de fonctions externes : externc.epi
    - externc.c, externc.h, externc_types.h

```
open Scheduler_data

fun deadline_miss_log(date:int;task_id:int) returns ()
fun random(max:int) returns (v:int)
fun print_scheduler_state(s:scheduler_state) returns ()
```

  - Algorithmes : scheduler.ept
  - main.c
  - Makefile ou compile.sh

Ordonnanceur

- Fonctions externes :
  - *random(max:int)* – retourne un entier entre 1 et max (y compris max et 1), en utilisant rand() (stdlib.h).
  - *deadline_miss_log(date,task_id)* – imprime un message d'erreur
  - *print_scheduler_state(s:scheduler_state)* – imprime un état d'ordonnancement. A utiliser dans:

```
node main() returns ()
var sstate, new_sstate : scheduler_state ;
let
  new_sstate = scheduler(sstate) ;
  sstate = init_sstate fby new_sstate ;
  () = print_scheduler_state(new_sstate) ;
tel
```

Objectif 1 – ordonnanceur RM

- Mise en œuvre du simulateur d'ordonnancements temps réel
  - Fonction main (C)
    - Maintenir un compteur de cycles, imprimé à chaque itération de la boucle infinie
    - Passage des cycles pas « Enter » (utilisation de fgets)
  - La configuration des transparents 12 et 19

Objectif 2 – ordonnanceur RM

- Modification de la configuration pour passer en celle du transparent 15
  - Non-ordonnançable – événement d'échéance manquée à détecter
  - Utiliser pour start_inst la version du transparent 34 (sans random)

Objectif 3 – ordonnanceur EDF

- Modification de la politique d'ordonnancement de RM à EDF
  - Configuration du transparent 15