Laurent MORIN
DESS Informatique
Promotion 2002 - 2003

RAPPORT DE PROJET

« Implantation du calcul de temps de réponse »

Abstract:

The aim of this report is to present a short bibliography and results of my project about response-time computation in real-time scheduling. The first chapter deals with problematic of real-time scheduling, presenting Quality of services constraints. The second chapter sum' up few results about classical static and dynamic priority scheduling algorithms.

The last two chapters are closer to this project, presenting Cheddar tool and my developments. Cheddar is a real-time scheduling simulator. My work consisted in implementing feasibility algorithms for classical algorithms like Rate-Monotonic or Earliest Deadline First. And to propose an algorithm to compute response-time observed during a scheduling simulation.

Keywords: Real-Time scheduling, Response-time computation, scheduling simulation analysis.

Résumé:

Ce rapport présente dans un premier temps une courte bibliographie permettant non seulement de bien cadrer la problématique de ce projet mais aussi de définir les concepts et d'énoncer les théorèmes qui seront utilisés par la suite.

La deuxième partie de ce rapport (chapitres 3 et 4) propose tout d'abord une présentation du simulateur d'ordonnancement temps-réel Cheddar, puis présente et explique les travaux que j'ai réalisés. Mes développements portent sur l'implémentation du calcul de temps de réponse par simulation d'une part, et sur l'étude de faisabilité pour les algorithmes classiques d'autre part.

Mots Clés: Ordonnancement temps-réel, Temps de réponse, Analyse de simulation

TABLE DES MATIERES

TABLE DES FIGURES

Chaptitre 1 - Qualité de service (QoS)

[I] Introduction

Le terme de qualité de service, même s'il est largement répandu dans la littérature, reste assez flou car il est utilisé pour qualifier un large spectre de domaines de l'informatique. Ainsi, on le retrouvera dans l'étude des systèmes temps réels critiques, des systèmes distribués ou encore dans le domaine des applications multimédia. Les notions sous-jacentes de ce terme sont intimement liées au sujet de ce projet, c'est pourquoi il est important de bien le définir pour comprendre le contexte général.

[II] La qualité de service dans les applications multimédia

Dans le cadre des applications multimédia, le terme de qualités de service désigne en général la résolution d'affichage d'une image ou d'une animation, la qualité d'un échantillon sonore, on peut encore retrouver le taux de compression…

[III] La qualité de service dans les systèmes distribués

Dans le cadre des systèmes répartis, on entend en général par le terme de qualité de service, la capacité du système à acheminer des informations d'une entité du système à une autre dans les délais attendus et avec un taux d'erreurs correct.

[IV] La qualité de service dans les systèmes temps-réel

Dans les systèmes dits temps-réel, le terme de qualité de service tend à désigner la capacité du système à respecter des contraintes temporelles plus ou moins fortes. On retrouvera ainsi des caractéristiques telles que la prédictibilité, l'optimalité du temps de réponse… La prise en compte et la gestion de la qualité de service dans de tels systèmes peut passer par l'implémentation de fonctions algorithmiques ou bien par une restructuration architecturale. La portée de ce projet se limite à l'aspect algorithmique.

Chaptitre 2 - Etat de l'art

[I] Modèle de tâches

Nous présentons dans ce paragraphe la définition du modèle de tâche. Ce travail est nécessaire pour fixer les notations utilisées et bien comprendre par la suite les objets et grandeurs manipulés.

Une tâche est une suite séquentielle d'occurrences. Chacune de ces occurrence occupe l'unité de traitement (généralement un processeur) pendant un temps noté C (on parlera alors de la charge d'une occurrence). Une contrainte de temps sur une tâche est donnée par une échéance relative notée D. L'échéance relative d'une tâche est la donnée d'une date à ne pas dépasser après le réveil d'une occurrence de la tâche. On définit alors l'échéance absolue par t+D ou t est l'instant d'arrivée d'une occurrence.

Lorsqu'une tâche ne comporte qu'une occurrence, elle est dite apériodique. Si les occurrences d'une même tâche sont espacée par une constante T (période de la tâche) la tâche sera dite périodique.

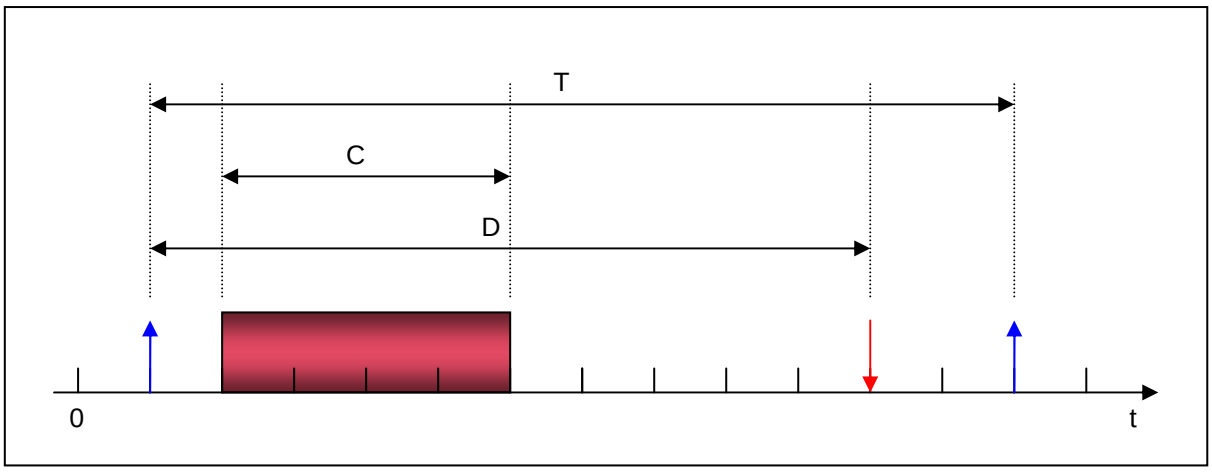

Figure 1 : Modèle simple de tâche

Une occurrence est l'entité élémentaire manipulée par un ordonnanceur [1]. Elle est caractérisée par un instant d'arrivée et un paramètre $E_{i,p}$ appelé donnée d'entrée utilisée par l'ordonnanceur. Les entiers i et p désignent la $p^{ième}$ occurrence de la tâche i. L'ensemble dans lequel $E_{i,p}$ prend ses valeurs est à fixer au départ. Un scénario d'occurrence d'une tâche est la donnée d'une suite croissante d'instants d'arrivée, notée $(t_p)_{p\in\aleph}$.

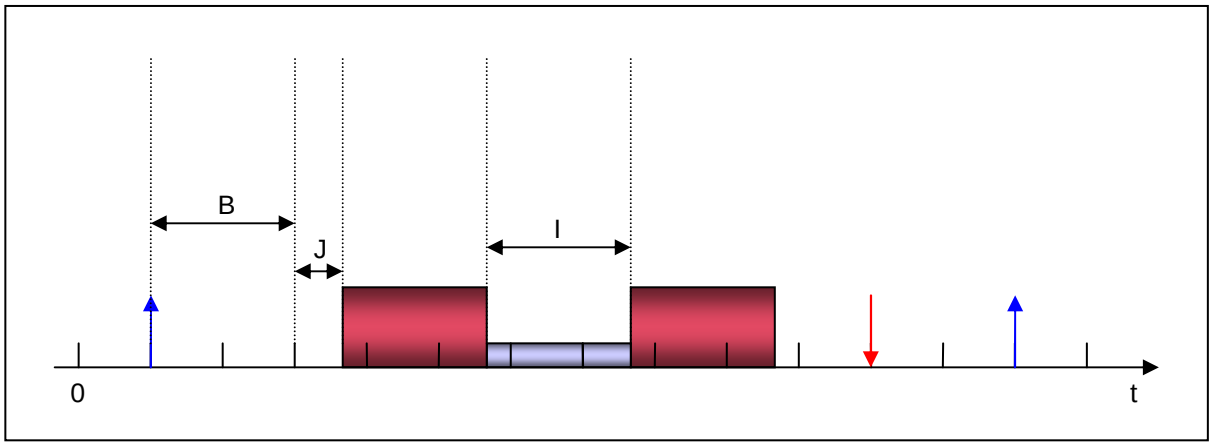

Figure 2 : Modèle complet de tâche

Afin d'assouplir le modèle de tâche [1] [2], on introduit un temps d'instabilité au réveil (Jitter Time) qu'on peut aussi appeler gigue d'arrivée. Ce paramètre sera noté J. Un autre paramètre noté B introduit la notion de temps de blocage. La laxité à l'instant t est définie par $L_i(t)=D_i-t-C_{i\ restant\ à\ exécuter}$. Enfin nous introduisons I comme l'interférence causée par une ou plusieurs autres tâche. L'interférence d'une tâche sur une autre est le temps maximum qu'elle pendant lequel elle peut là préempter.

L'ordonnancement temps réel permet la planification de l'attribution des unités de traitement aux tâches au cours du temps. Le problème étant de pouvoir disposer de preuves de la faisabilité à priori. Un Ordonnancement sera dit faisable si toutes les contraintes de temps sont respectées. On comprend assez bien que pour des systèmes critiques mettant en jeu des vies humaines, il n'est pas possible de se contenter de résultats de tests pour la validation.

Dans sa thèse, Laurent Leboucher [1] pause des critères permettant d'évaluer la faisabilité pour certains algorithmes. Nous allons donc présenter dans ce paragraphe les différents algorithmes sur lesquels porte ce projet. Pour chacun d'entre eux, deux variantes existent suivant le caractère préemptif ou non du système considéré.

A - Rate Monotonic (RM)

Rate Monotonic fait partie de la famille des algorithmes à priorité fixe. Le principe consiste à affecter aux tâches des priorités qui sont inversement proportionnelles à leurs périodes. Au moment du choix d'une tâche à exécuter, la prochaine tâche élue est celle ayant la plus forte priorité. Les tâches à ordonnancer doivent être périodique et à échéance sur requête.

B - Deadline Monotonic (DM)

Deadline Monotonic est un algorithme à priorité fixe proche de Rate Monotonic à ceci près qu'il affecte à une tâche une priorité inversement proportionnelle au délai critique.

C - Earliest Deadline First (EDF)

Earliest Deadline First est un ordonnancement à priorité dynamique. Une tâche est d'autant plus prioritaire que sa date d'échéance absolue est proche de la date courante.

D - Least Laxity First (LLF)

Least Laxity First est un ordonnancement à priorité dynamique. Les tâches sont d'autant plus prioritaires que leur laxité est faible à la date courante.

[II] Résultats généraux pour le calcul du temps de réponse

A - Introduction

Nous présentons dans cette partie, les résultats généraux sur le temps de réponse au pire cas pour les algorithmes les plus communs. L'intérêt d'un tel calcul est de donner d'une part une condition nécessaire et suffisante de faisabilité mais aussi de quantifier la qualité de service fournie par le système pour chacune des tâches.

B - Algorithmes à priorités fixes (RM/DM)

Principe :

Le temps de réponse d'une tâche (selon le modèle simple) est calculé à partir de la formule suivante :

$$r_i = C_i + I_i$$

où I_i désigne le temps maximum d'interférence sur la tâche i.

Dans le modèle simple d'une tâche, Le calcul de l'interférence d'une tâche τ_j sur une tâche τ_i revient en fait à déterminer le nombre maximum n_o d'occurrence de τ_j qui peuvent intervenir pendant le temps r_i . Soit T_j la période de τ_j, on obtient ainsi $n_o = \left\lceil \frac{r_i}{T_j} \right\rceil$ ce qui nous donne un temps maximum d'interférence $I_{j \to i} = \left\lceil \frac{r_i}{T_j} \right\rceil \times C_i$

Du coup l'interférence totale sur la tâche τ_i est donnée par $I_i = \sum_{j \in hp(i)} \left\lceil \frac{r_i}{T_j} \right\rceil \times C_i$

En réinjectant le calcul de I_i dans la formule de r_i, on obtient une équation fonction de r_i. Le théorème suivant propose une méthode de calcul par des approximations successives convergentes.

Théorème (Lechoczky 1990) : *Le temps de réponse au pire cas pour un jeu de tâches est obtenu lorsque toutes les tâches ont leur fréquence d'arrivée maximale et arrivent au même instant critique t=0. On peut calculer r_i (le temps de réponse de la tâche i) grâce à l'équation suivante :*

$$\begin{cases} r_i = \max_{0 \le q \le Q} (w_{i,q} - qT_i) \\ \text{où Q est le plus petit entier tel que } w_{i,Q} \le (Q+1)T_i \text{ et :} \\ w_{i,q}^{k+1} = (q+1)C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_{i,q}^k}{T_j} \right\rceil \times C_j \\ \text{où hp(i) désigne l'ensemble des indice des tâches de priorité supérieur ou égale à i.} \end{cases}$$

Les résultats que nous allons énoncés dans les points suivants, tirés des travaux de Tindell, Burn et Wellings, adaptent le théorème de Lechoczky au modèle complet d'une tâche.

(a) Cas préemptif

$$\begin{cases} r_i = \max_{0 \le q \le Q} (w_{i,q} + J_i - qT_i) \\ \text{où Q est le plus petit entier tel que } w_{i,Q} \le (Q+1)T_i \text{ et :} \\ w_{i,q} = (q+1)C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_{i,q} + J_j}{T_j} \right\rceil \times C_j \end{cases}$$

(b) Cas non préemptif

$$r_i = \max_{0 \le q \le Q}(w_{i,q} + C_i + J_i - qT_i)$$

où Q est le plus petit entier tel que $w_{i,Q} + C_i \le (Q+1)T_i$ *et :*

$$w_{i,q} = (q+1)C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_{i,q} + J_j}{T_j} \right\rceil \times C$$

C - Algorithmes à priorité dynamiques (EDF/LLF)

(a) Définitions

L_i (a) est la longueur de la longueur de la période occupée qui inclut l'occurrence de τ_i arrivée à l'instant a.

W_i (a,t) correspond au nombre d'occurrences de tâches τ_j (avec $j \neq i$) arrivées dans l'intervalle [0 ; t [et qui ont une échéance absolue inférieure ou égale à $a+D_i$.

(b) Cas préemptif

Théorème (Spuri 1996) : *Le temps de réponse au pire cas pour un jeu de tâches est obtenu lorsque toutes les tâches ont leur fréquence d'arrivée maximale et arrivent, sauf éventuellement l'instance de la tâche considérée, au même instant critique t=0. On peut calculer* r_i *(le temps de réponse de la tâche i grâce à l'équation suivante :*

$$r_i = \max_{a \in S}(L_i(a) - a) \text{ avec}$$

$$L_i(a) = W(a, L_i(a)) + \left(1 + \left\lfloor \frac{a}{T_i} \right\rfloor\right) C_i \quad \text{(plus petit point fixe racine de l'équation)}$$

$$S = \bigcup_{j=1}^{n} \left\{ kT_j + D_j - D_i, 0 \le k \le \left\lfloor \frac{\min(\lambda, L_i)}{T_j} \right\rfloor \right\}$$

$$\lambda = \sum_{j=1}^{n} \left\lceil \frac{\lambda}{T_j} \right\rceil C_j \quad \text{(première racine de cette équation)}$$

L_i (La plus grande période à échéance pour la tâche τ_i)

(c) Cas non préemptif

Théorème (George, Rivierre et Spuri 1996) : *Soit un référentiel* $\sum = (\Upsilon, \Pi)$ *où* Υ *contient tout trafic périodique non concret non préemptif et* Π *contient EDF.*

$$r_i = \max_{a \in S}\{C_i, L_i(a) - a\} \text{ avec}$$

$$L_i(a) = \max_{D_j > a + D_i}\{C_j\} + \sum_{j \neq i,\, D_j \leq a + D_i} \min\left\{1 + \left\lfloor \frac{L_i(a)}{T_j} \right\rfloor, 1 + \left\lfloor \frac{a + D_i - D_j}{T_j} \right\rfloor\right\} C_j + \left\lfloor \frac{a}{T_i} \right\rfloor C_i$$

$$S = \bigcup_{j=1}^{n} \left\{ kT_j + D_j - D_i, 0 \leq k \leq \left\lfloor \frac{\min(L, L_i)}{T_j} \right\rfloor \right\}$$

$$\lambda = \sum_{j=1}^{n} \left\lceil \frac{\lambda}{T_j} \right\rceil C_j \quad \text{(première racine de cette équation)}$$

L_i (La plus grande période à échéance pour la tâche τ_i)

Chaptitre 3 - Cheddar

[I] Présentation

Cheddar est un simulateur d'ordonnancement temps réel à vocation éducative diffusé sous licence GNU/GPL. Il est développé au sein de l'équipe LIMI de l'université de Brest. L'outil, écrit en Ada et Gtk/Ada, est portable sur des plates-formes Win32, Solaris et Linux.

[II] Présentation de l'outils Cheddar

Cheddar dans sa version actuelle, permet de visualiser des ordonnancements temps réels classiques (RM/DM/EDF/LLF). Le calcul du taux d'utilisation d'un processeur fournit une condition nécessaire pour la faisabilité d'un ordonnancement. Le calcul du temps de réponse n'est implanté que

pour RM et DM dans le cas préemptif. Cheddar supporte des tâches périodiques et apériodiques. Il prend également en compte le partage de ressources grâce aux protocoles PIP et PCP. Cheddar fournit également des outils pour l'expression et la simulation de tâches comportant des relations de dépendance. Enfin, un framework permet de définir et tester des ordonnanceurs.

Chaptitre 4 - Travail demandé

[I] Les Objectifs du projet

- Modifier le calcul du temps de réponse existant pour Rate Monotonic et Deadline Monotonic dans le cas préemptif pour l'adapter au cas non préemptif.

- Définir et implanter un algorithme de calcul de temps de réponse par simulation. Uniformiser l'affichage des résultats avec celui du calcul théorique.

- Implémenter les algorithmes de calcul de temps de réponse pour Earliest Deadline First et Least Laxity First.

[II] Calcul du temps de réponse par simulation

A - Présentation

Le calcul par simulation m'a été demandé pour plusieurs raisons :

D'une part, les formules du chapitre 2 concernant les algorithmes à priorité dynamiques n'ont, semble-t-il, jamais fait l'objet de vérifications expérimentales. Ainsi, le fait d'implémenter ces formules dans Cheddar et de pouvoir confronter leurs résultats avec ceux obtenus par simulation, permet de montrer qu'il y a une erreur soit dans l'implémentation des formules, soit dans les formules elles même. Loin de prouver l'exactitude de ces formules, Cheddar pourra (après une validation sérieuse du calcul par simulation) au moins prouver si elles sont fausses. En effet, il paraît évident qu'un temps de réponse par simulation ne peut en aucun cas être supérieur à un temps de réponse au pire cas calculé.

D'autre part, Cheddar offrant la possibilité de définir ses propres ordonnanceurs pour lesquels il n'existe évidemment pas de formules magiques, il peut être intéressant grâce à cet outils de générer aléatoirement un grand nombre de jeux de tâches et d'observer les temps de réponse expérimentaux. Ceci pouvant être utile pour l'élaboration de formules de faisabilité.

B - L'algorithme

(a) Principe

J'ai du, pour implémenter ma solution, étendre le framework de Cheddar afin de modéliser la notion d'occurrence de tâche.

Une séquence d'ordonnancement est le résultat produit par Cheddar pour une simulation. Chaque item de cette séquence est un doublet <date, tâche> représentant pour chaque date la tâche en cours d'exécution sur le processeur.

Le schéma d'occurrence est définit au départ. Sa taille est bornée par $\frac{\Delta}{T_i}+1$ (ou Δ est la longueur de l'intervalle d'étude. Il présente une séquence ordonnée d'occurrences d'une tâche. Pour chaque occurrence, on est capable de prédire sa date d'arrivée en fonction de son rang dans le schéma d'occurrence ($k \times T_i$) et son échéance absolue ($k \times T_i + D_i$) La capacité résiduelle (initialisée à la capacité de la tâche) est décrémentée à chaque fois que l'occurrence a la main sur le processeur. Tandis que le temps de réponse de l'occurrence est incrémenté tant que celle ci est active. L'occurrence deviendra inactive lorsque la capacité résiduelle sera nulle.

(b) Code

L'algorithme suivant correspond au calcul pour une tâche étudiée périodique.

```
procedure Check_Temporal_Constraints (

  My_Task : in     Periodic_Task_Ptr;
  Sched   : in     Scheduling_Sequence;
  Result  : in out Unbounded_String     ) is

  Max_Occurence_Number             : Natural := 0;
  Awoken_Task_Occurence_Number     : Natural := 0;
  Running_Awoken_Task_Index        : Natural := 0;
  Next_My_Task_Occurence_Awake     : Natural := 0;
  My_Task_Worst_Case_Response_Time : Natural := 0;
  Running_Task                     : Task_Ptr;
  Effective_End                    : Natural := 0;
  Deadline_Message         : Unbounded_String := To_Unbounded_String("") ;
  Current_Date                     : Natural := 0;


begin

-- déterminer le nombre maximum d'occurrence dans l'intervalle d'étude
  Max_Occurence_Number :=
     Sched.Scheduled.Entries(Sched.Scheduled.Nb_Entries - 1).Data
     / My_Task.Period + 1 ;
```

```
  declare

     My_Task_Occurence_Table:Task_Occurence_Table(0..Max_Occurence_Number);

  begin

         -- initialisation de toutes les occurences
         for I in 0..Max_Occurence_Number loop
            Initialize(My_Task_Occurence_Table(I) , My_Task , I) ;
         end loop ;

         -- pour chaque date donnée dans la séquence d'ordonnancement
         for T in 0 .. Sched.Scheduled.Nb_Entries-1 loop

            -- la date courante
            Current_Date := Sched.Scheduled.Entries(T).Data ;

            -- la tâche occupant le processeur à la date courrante
            Running_Task := Sched.Scheduled.Entries(T).Item ;

            -- détecter le réveil d'une occurrence de la tâche étudiée
            if ( Current_Date =
My_Task_Occurence_Table(Next_My_Task_Occurence_Awake).Arrival_Date )
            then

              -- une occurrence de plus est réveillée
              Awoken_Task_Occurence_Number:=Awoken_Task_Occurence_Number+1;

              -- activer l'occurrence réveillée
            My_Task_Occurence_Table(Next_My_Task_Occurence_Awake).Is_Active
            := True ;

              -- s'il n'y avait pas d'occurrence active alors la nouvelle
              -- occurrence sera celle en cours d'exécution
              if ( Awoken_Task_Occurence_Number = 0 )
              then
                  Running_Awoken_Task_Index:=Next_My_Task_Occurence_Awake;
              end if ;

              -- la prochaine occurrence à réveiller
              Next_My_Task_Occurence_Awake:=Next_My_Task_Occurence_Awake+1;

            end if ;

            -- s'il y a des occurrences actives
            if ( Awoken_Task_Occurence_Number > 0 )
            then

               -- incrementer le temps de réponse de toutes les
               -- occurrences actives
               for I_Occurence in 0 .. Max_Occurence_Number loop
                  if ( My_Task_Occurence_Table(I_Occurence).Is_Active )
                  then
                     My_Task_Occurence_Table(I_Occurence).Response_Time :=
                       My_Task_Occurence_Table(I_Occurence).Response_Time
                       + 1 ;
                  end if ;
               end loop ;
```

```
               -- dans le cas ou la tâche en cours d'exécution correspond à
               -- la tâche étudiée
               if ( Running_Task.Name = My_Task.Name )
               then

-- décrémenter la capacité résiduelle de l'oocurence en cours d'exécution
My_Task_Occurence_Table(Running_Awoken_Task_Index).Residual_Capacity :=
My_Task_Occurence_Table(Running_Awoken_Task_Index).Residual_Capacity
- 1 ;

                  -- si cette occurrence est terminée
                  if
(My_Task_Occurence_Table(Running_Awoken_Task_Index).Residual_Capacity = 0)
                  then
                     Effective_End := Natural(Current_Date) + 1 ;

                     -- verification de l'échéance de occurrence
                     if
( My_Task_Occurence_Table(Running_Awoken_Task_Index).Response_Time
                          > My_Task.Deadline )
                     then
                        Deadline_Message := Deadline_Message
                          & Lb_Check_Deadline(Current_Language)
& My_Task_Occurence_Table(Running_Awoken_Task_Index).Absolute_Deadline'Img
                          & Lb_Completion_Time(Current_Language)
                          & Effective_End'Img
                          & To_Unbounded_String(")");

                     end if ;

                     -- mise à jour du pire temps de réponse
                     -- pour la tâche étudiée
                     if
(My_Task_Occurence_Table(Running_Awoken_Task_Index).Response_Time
                         > My_Task_Worst_Case_Response_Time)
                     then
                        My_Task_Worst_Case_Response_Time :=

My_Task_Occurence_Table(Running_Awoken_Task_Index).Response_Time ;
                     end if ;

                     -- désactiver l'occurrence terminée
My_Task_Occurence_Table(Running_Awoken_Task_Index).Is_Active := False ;

                     -- passer la main à la prochaine occurrence
                     Running_Awoken_Task_Index := Running_Awoken_Task_Index
+ 1 ;

                     -- une occurrence de moins réveillée
                     Awoken_Task_Occurence_Number :=
Awoken_Task_Occurence_Number - 1 ;

                  end if ;

               end if ;

            end if ;

         end loop ;
```

```
         Result := Result
           & To_Unbounded_String("    ")
           & To_String(My_Task.Name)
           & To_Unbounded_String(" => ")
           & Format(My_Task_Worst_Case_Response_Time)
           & Deadline_Message
           & Unbounded_Lf ;

      end ;

end Check_Temporal_Constraints;
```

L'algorithme suivant correspond au calcul pour une tâche étudiée apériodique. Celui-ci est nettement plus simple car il ne considère qu'une seule occurrence.

```
procedure Check_Temporal_Constraints (
  My_Task : in     Aperiodic_Task_Ptr;
  Sched   : in     Scheduling_Sequence;
  Result  : in out Unbounded_String     ) is

  My_Task_Occurence : Task_Occurence ;
  Running_Task      : Task_Ptr ;
  Effective_End     : Natural := 0;
  Deadline_Message  : Unbounded_String := To_Unbounded_String("") ;
  Current_Date      : Natural := 0;
begin

  -- initialiser l'occurrence unique
  Initialize(My_Task_Occurence , My_Task ) ;

    -- pour chaque item de la séquence
    for T in 0 .. Sched.Scheduled.Nb_Entries-1 loop

      -- la date courante
      Current_Date := Sched.Scheduled.Entries(T).Data ;

      -- la tâche occupant le processeur à la date courrante
      Running_Task := Sched.Scheduled.Entries(T).Item ;

      -- si l'occurrence doit être réveilée
      if ( Current_Date = My_Task_Occurence.Arrival_Date ) then
        My_Task_Occurence.Is_Active := True ;
      end if ;

      -- incrementer le temps de réponse de l'occurrence
      -- si elle est active
      if ( My_Task_Occurence.Is_Active ) then
        My_Task_Occurence.Response_Time :=
          My_Task_Occurence.Response_Time + 1 ;
      end if ;

      -- dans le cas ou la tâche en cours d'exécution correspond à
      -- la tâche étudiée
      if ( Running_Task.Name = My_Task.Name )
      then

        My_Task_Occurence.Residual_Capacity :=
```

```
          My_Task_Occurence.Residual_Capacity - 1 ;

        -- si l'occurrence est terminée
        if (My_Task_Occurence.Residual_Capacity = 0)
        then
          My_Task_Occurence.Is_Active := False ;

          Effective_End := Current_Date + 1 ;

          -- verification de l'échéance de occurrence
          if ( My_Task_Occurence.Response_Time > My_Task.Deadline ) then
            Deadline_Message := Deadline_Message
                    & Lb_Check_Deadline(Current_Language)
                    & My_Task_Occurence.Absolute_Deadline'Img
                    & Lb_Completion_Time(Current_Language)
                    & Effective_End'Img
                    & To_Unbounded_String(")");
        end if ;
      end if;
    end if;
  end loop;

      Result := Result
        & To_Unbounded_String("    ")
        & To_String(My_Task.Name)
        & To_Unbounded_String(" => ")
        & Format(My_Task_Occurence.Response_Time)
        & Deadline_Message
        & Unbounded_Lf ;

end Check_Temporal_Constraints;
```

(c) Remarques

Cet algorithme pourrait ne pas fonctionner correctement si l'Ordonnanceur autorise des temps non occupés avec des occurrences en attente. En effet ces instants n'apparaîtraient pas dans la séquence d'ordonnancement. Pour corriger ou du moins détecter ce genre de cas, il suffit de calculer le temps de réponse une fois l'occurrence terminée (en faisant la soustraction entre la date de terminaison et la date d'arrivée) et non pas de l'incrémenter au fur et à mesure.

Le temps de réponse de la dernière occurrence ne sera pas complet car il manquera l'information de terminaison. Pour éviter ceci, il suffit d'augmenter un peu l'intervalle de temps de la simulation.

Des exemples de résultats par simulation apparaîtront dans la suite de ce rapport avec les exemples pour RM/DM et EDF/LLF.

[III] Calcul du temps de réponse pour RM/DM

A - Présentation

Dans un premier temps, je me suis penché sur l'algorithme qui était déjà implanté. Celui-ci était une adaptation du théorème énoncé dans le Chapitre 2 de ce rapport. Afin de favoriser la compréhension du codage de l'algorithme, j'ai dans un premier temps retravaillé les fonctions pour qu'elles soient plus proches du théorème.

Outre une meilleure lisibilité de l'algorithme, ce travail m'a permit de mettre en évidence et de corriger une erreur dans le calcul de la charge de travail ($w_{i,q}$).

Le calcul du temps de réponse d'une tache est effectué après avoir vérifier les préconditions propres à chaque algorithme d'ordonnancement. Ensuite, une première procédure permet de calculer, pour un indice q donné, la charge de travail. Cette dernière étant utilisé pour calculer le temps de réponse.

B - L'algorithme

(a) Charge de travail

Dans le théorème, l'expression de la charge de travail ($w_{i,q}$) pose problème car elle apparaît dans les parties gauche et droite de l'expression. Une manière de calculer cette expression consiste à calculer des approximations successives et de s'arrêter lorsqu'il y a convergence. Nous retrouverons ce type de calcul de point fixe très souvent par la suite.

$$w_{i,q}^{k+1} = (q+1)C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_{i,q}^{k} + J_j}{T_j} \right\rceil \times C_j$$

```
function Compute_Workload (
             My_Tasks : in     Tasks_Set;  -- l'ensemble des tâches
             Taski    : in     Task_Ptr;   -- la tâche étudiée
             q        : in     Integer     -- la valeure de q considérée
             ) return Double is

   Iterator  : Tasks_Iterator;
   Taskj     : Task_Ptr;
   Wiq_k,                  -- k^ième approximation de Wiq
   Wiq_k1    : Double;     -- k+1^ième approximation de Wiq

begin

   -- initialisation avec des valeurs différentes
   -- pour qu'il y ait au moins une itération
   Wiq_k  := -0.1 ;
   Wiq_k1 :=  0.0 ;

   -- tant qu'il n'y a pas convergences des approximations
```

```
      while Wiq_k /= Wiq_k1  loop

         Reset_Iterator(Iterator);

         Wiq_k  := Wiq_k1 ;
         Wiq_k1 := Double((q+1)*Taski.Capacity)
                 + Double(Taski.Blocking_Time) ;

         -- pour chacune des tâches de plus grande priorité
         loop
            Current_Element(My_Tasks, Taskj, Iterator);

            if(Taskj.Priority> Taski.Priority)
            then
               Wiq_k1 :=   Wiq_k1
                 + Double'Ceiling((Wiq_k +
                                   Double(Periodic_Task_Ptr(Taskj).Jitter))
                 / Double(Periodic_Task_Ptr(Taskj).Period))
                 * Double(Taskj.Capacity) ;
            end if ;

            exit when Is_Last_Element(My_Tasks, Iterator2) = True;

            Next_Element(My_Tasks, Iterator);

         end loop ;

      end loop ;

      return Wiq_k1;

   end Compute_Workload ;
```

(b) Temps de réponse

L'algorithme s'appuie sur le théorème du chapitre 2.

```
   procedure Compute_Response_Time (
                My_Scheduler   : in     Fixed_Priority_Scheduler;
                      My_Tasks       : in out Tasks_Set;
                Processor_Name : in     Unbounded_String;
                Msg            : in out Unbounded_String;
                Response_Time  :    out Response_Table ) is


      Tmp       : Tasks_Set;
      Iterator  : Tasks_Iterator;
      Taski     : Task_Ptr;
      i         : Response_Range;
      Wiq,
      Ri        : Double         := 0.0;
      q         : Integer;

   begin
      -- initialisation des temps de réponse
      Double_Tasks_Parameters.Initialize(Response_Time);

      -- sélectionner uniquement les tâches du processeur voulu
      Current_Processor_Name:=Processor_Name;
      Select_And_Copy( My_Tasks, Tmp, Select_Cpu'access );

      -- Positionner les priorités en fonction l'ordonnanceur
      if (My_Scheduler.Name = Deadline_Monotonic)
      then
         Set_Priority_According_To_Dm(My_Scheduler, Tmp);
```

```
      else
         if (My_Scheduler.Name = Rate_Monotonic)
         then
            Set_Priority_According_To_Rm(My_Scheduler, Tmp);
         end if;
      end if;

      -- Ordonner les tâches par ordre de priorité
      Sort(Tmp, Increasing_Priority'access);

      Reset_Iterator(Iterator);

      -- index de la tâche courrante
      i := 0 ;

       -- calcul du temps de réponse de toutes les tâches
      loop

         -- sélection de la tâche courante
         Current_Element(Tmp, Taski, Iterator);

         -- initialiser le temps de réponse pour la tâche courante
         Response_Time.Entries(i).Data := 0.0;
         Response_Time.Entries(i).Item := Taski;
         Response_Time.Nb_Entries      := Response_Time.Nb_Entries+1;

         -- trouver le plus grand Ri dans l'intervalle [0..Q]
         q := 0 ;
         loop

            Wiq := Compute_Workload(Tmp, Taski, q);

            -- Ri = Wiq + Ji - qTi
            Ri :=   Wiq
              + Double(Periodic_Task_Ptr(Taski).Jitter)
              - Double(q) * Double(Periodic_Task_Ptr(Taski).Period) ;

            -- doit-on faire une mise à jour du maximum pour Ri
            if ( Ri > Response_Time.Entries(i).Data )
            then
               Response_Time.Entries(i).Data := Ri ;
            end if ;

            -- sortie de boucle lorsque Q est trouvé ( WiQ<=(Q+1)Ti )
            exit when Wiq <= Double(q+1) * Double(Periodic_Task_Ptr(Taski).Period);

            q := q + 1 ;

         end loop ;


         -- dans le cas où l'ordonnanceur est non préemptif,
          -- il suffit de rajouter Ci à Ri
         if ( My_Scheduler.Is_Preemptive = Non_Preemptive )
         then
            Response_Time.Entries(i).Data := Response_Time.Entries(i).Data +
Double(Periodic_Task_Ptr(Taski).Capacity) ;
         end if ;

         exit when Is_Last_Element(tmp, Iterator) = True;

         Next_Element(tmp, Iterator);
         i:=i+1 ;

      end loop ;

   end Compute_Response_Time ;
```

[IV] Calcul du temps de réponse pour EDF/LLF

A - Présentation

Ce calcul présente le même genre de difficulté que pour RM/DM. En effet, les expressions données par le théorème du chapitre 2 sont des équations de point fixe. Ainsi, nous retrouvons le principe de calcul par approximations successives grâce à une équation récursive.

Le problème majeur de ce calcul est de déterminé l'ensemble S sur lequel on doit effectuer les calculs. La formule énoncée dans le chapitre 2 n'est pas applicable car le calcul utilise les priorités des tâches. Un calcul fin de S permet d'effectuer moins de calcul, toutefois, les résultats obtenus pour les exemples que j'ai traités sont globalement satisfaisants pour l'intervalle S définit par :

$S = \bigcup_{j=1}^{n} \{kT_j + D_j - D_i, k \in \aleph\} \cap [0, L_i]$ (où L_i est le plus petit point de convergence entre a et $L_i(a)$.

L'ensemble S est un ensemble de valeurs entières non continu et le stockage de ces valeurs ne serait pas chose aisée en Ada. C'est pourquoi le calcul se fait « à la volée » afin d'éviter le stockage. Cette manière de procéder permet en outre d'alléger la complexité de l'algorithme.

B - L'algorithme

Les trois fonctions suivantes permettent de calculer le temps de réponse au pire cas pour un jeu de tâche donné.

```
function Compute_Wi_A_t(
  My_Tasks       : in Tasks_Set;
  Taski          : in Task_Ptr;
  A              : in Integer;
  T              : in Double
) return Double is

  Result    : Double :=0.0;
  Min       : Double :=0.0;
  Iterator  : Tasks_Iterator;
  Taskj     : Task_Ptr;

begin

  Reset_Iterator(Iterator);
  loop
    Current_Element(My_Tasks, Taskj, Iterator);

    if ((Taskj.Name /= Taski.Name)
```

```
         and (Taskj.Deadline <= A + Taski.Deadline) )
    then
      Min := 1.0
             + Double'Floor(
                     (Double(A) + Double(Taski.Deadline)
                      - Double(Taskj.Deadline))
             / Double(Periodic_Task_Ptr(Taskj).Period)) ;

      if (Double'Ceiling(T/Double(Periodic_Task_Ptr(Taskj).Period)) < Min )
      then
        Min := Double'Ceiling(T/Double(Periodic_Task_Ptr(Taskj).Period));
      end if ;

      Result := Result + Min * Double(Taskj.Capacity) ;
    end if;

    exit when Is_Last_Element(My_Tasks, Iterator) = True;
    Next_Element(My_Tasks, Iterator);
  end loop ;

  return Result ;
end Compute_Wi_A_t ;
```

```
function Compute_Li_a (
  My_Scheduler   : in     dynamic_priority_Scheduler;
  My_Tasks       : in Tasks_Set;
  Taski          : in Task_Ptr;
  A              : in Integer
) return Double is

  Li_A      : Double :=0.1;
  Li_A_1    : Double :=0.0;
  Iterator  : Tasks_Iterator;
  Taskj     : Task_Ptr;
  Max_Cj    : Natural:=0 ;
  Tmp       : Double:=0.0 ;
  Min       : Double:=0.0 ;
  Min1      : Double:=0.0 ;
  Min2      : Double:=0.0 ;

begin

  if ( My_Scheduler.Is_Preemptive = Preemptive )
  then
    -- cas préemptif
    while Li_A_1 /= Li_A loop
      Li_A := Li_A_1 ;
      Li_A_1 := Compute_Wi_A_t(My_Tasks, Taski, A, Li_A) ;
      Li_A_1 := Li_A_1
                + ( 1.0 + Double'Floor( Double(A) /
                          Double(Periodic_Task_Ptr(Taski).Period) ) )
                * Double(Taski.Capacity) ;
    end loop ;

  else
    -- cas non préemptif
    while Li_A_1 /= Li_A loop

    Li_A := Li_A_1 ;
```

```
   Tmp := 0.0 ;
   Reset_Iterator(Iterator);
   loop
     Current_Element(My_Tasks, Taskj, Iterator);
     if ( (A + taski.Deadline) < Taskj.Deadline )
     then
       if ( Taskj.Capacity > Max_Cj)
       then
         Max_Cj := Taskj.Capacity ;
       end if ;
     end if ;

     if ( (Taskj.Name /= Taski.Name)
          and (Taskj.Deadline <= A + Taski.Deadline) )
     then
       Min1 := 1.0
               + Double'Floor( Li_A
                         / Double(Periodic_Task_Ptr(Taskj).Period) ) ;

       Min2 := 1.0
               + Double'Floor(Double(A) + Double(Taski.Deadline)
               - Double(Taskj.Deadline))
               / Double(Periodic_Task_Ptr(Taskj).Period) ;

       Min := Min1 ;

       if (Min2 < Min)
       then
         Min := Min2 ;
       end if ;

       Tmp := Tmp + Min * Double(Taskj.Capacity)
              +  Double'Floor(Double(A)
              / Double(Periodic_Task_Ptr(Taskj).Period))
              * Double(Taski.Capacity) ;
     end if ;

     exit when Is_Last_Element(My_Tasks, Iterator) = True;
     Next_Element(My_Tasks, Iterator);
   end loop ;

   Li_A_1 := Double(Max_Cj) + Tmp ;
end loop ;

end if ;

     return Li_A ;
 end Compute_Li_A ;
```

```
  -- cette function peut presenter dans certains cas non-preemptifs
  -- Cela semble venire de la borne suppérieure de a:(exit when a>=Li(a))
  procedure Compute_Response_Time (
        My_Scheduler   : in     dynamic_priority_Scheduler;
        My_Tasks       : in out Tasks_Set;
        Processor_Name : in     Unbounded_String;
        Msg            : in out Unbounded_String;
```

```
      Response_Time  :    out Response_Table    ) is

   Tmp       : Tasks_Set;
   Iterator1 : Tasks_Iterator;
   Taski     : Task_Ptr;
   Iterator2 : Tasks_Iterator;
   Taskj     : Task_Ptr;
   I         : Response_Range :=0;

   A         : Integer := 0;

   B_Sup     : Double:=0.0;
   Li_a      : Double:=0.0;
   K         : Natural:=0;

begin
   Double_Tasks_Parameters.Initialize(Response_Time);

   -- check if tasks are periodics
   Periodic_Control(My_Tasks, Processor_Name);

   -- check start time
   Start_Time_Control(My_Tasks, Processor_Name);

   -- check offset
   Offset_Control(My_Tasks, Processor_Name);

   --
   Current_Processor_Name:=Processor_Name;
   Select_And_Copy( My_Tasks, Tmp, Select_Cpu'access );
   Reset_Iterator(Iterator1);


   loop
      -- selection of the current task
      Current_Element(Tmp, Taski, Iterator1);

      -- initialize response time for current task
      Response_Time.Entries(i).Data := 0.0;
      Response_Time.Entries(i).Item := Taski;
      Response_Time.Nb_Entries := Response_Time.Nb_Entries + 1;

      if ( My_Scheduler.Is_Preemptive = Preemptive )
      then
         Response_Time.Entries(i).Data := Double(Taski.Capacity) ;
      end if ;

      -- compute S "on the fly"
      Reset_Iterator(Iterator2);

      loop
         Current_Element(Tmp, Taskj, Iterator2);

         K := 0 ;
         loop
           A:= (K * Periodic_Task_Ptr(Taskj).Period)
             + Taskj.DeadLine
             - Taski.DeadLine ;

           Li_A := Compute_Li_A(My_Scheduler, Tmp, Taski, A) ;
```

```
            if ( (Li_A - Double(A) ) > Response_Time.Entries(i).Data )
            then
               Response_Time.Entries(i).Data := Li_A - Double(A) ;

               -- PROBLEM
               -- with non preemptive case response_time has no upper
               -- bound in some cases
               -- Put(Natural(Response_Time.Entries(i).Data)'Img) ;
            end if ;

            if ( My_Scheduler.Is_Preemptive = Preemptive )
            then
               B_Sup := Li_A ;
            else
               B_Sup := Li_A ;
            end if ;

            exit when (Double(A) >= B_Sup) ;
            K := K + 1 ;
         end loop ;

         exit when Is_Last_Element(tmp, Iterator2);
         -- go to the next task
         Next_Element(tmp, Iterator2);
      end loop ;

      exit when Is_Last_Element(tmp, Iterator1) = True;

      -- go to the next task
      Next_Element(tmp, Iterator1);

      I := I + 1 ;
   end loop ;

end Compute_Response_Time;
```

[V] Remarques générales

Le langage Ada étant réputé pour son typage fort, il me semble qu'il faudrait resserrer les types des variables plus de cohérence et peut être moins de chance de bug.

Plus de tests sont également nécessaires avant de valider l'ensemble des algorithmes implémenter.

Il semble qu'il y ait un problème pour le cas non préemptif de EDF/LLF il peut provenir soir d'une vérification manquante dans les préconditions soit d'une erreur dans la close de sortie de boucle.

Conclusion et remerciements

Ce projet m'a permis d'approfondir mes connaissances des ordonnancements temps réels. L'étude bibliographique sur l'influence de la gigue pour Rate Monotonic, réalisée pour le module SRI de maîtrise, m'a permis de rentrer plus facilement dans ce sujet et a facilité la compréhension.

Outre la compréhension des formules mathématiques et des notions mises en jeu, la barrière du langage (Ada) a été un réel frein au moins au départ. De plus, la prise en main du framework de l'outil Cheddar nécessite du temps.

Je tiens à remercier Frank Singhoff pour ce sujet et la qualité de son encadrement.

Références Bibliographiques

[1] Laurent Leboucher – « ***Algorithmique et modélisation pour la qualité de service des systèmes répartis temps réel*** » - ENST 98 E 018

[2] N. Audsley, A. Burns, M. Richardson, K. Tindell, A.J. Wellings – « ***Applying New Scheduling Theory to static Priority Pre-emptive Scheduling*** »

[3] L. George, N. Riviere, M. Spuri – « **Preemptive and non-preemptive real-time uniprocessor scheduling** » Inria Research Report 2966