Rapport de Projet

Développement de programmes de vérification d'ordonnancement temps réel

Master 2 Informatique et Systèmes complexes

Responsable : Frank SINGHOFF

Année : 2008-2009

Pingyong ZHANG

Table des matières

1. Introduction

Dans le cadre de la Master Informatique, il nous a été demandé d'effectuer un projet de fin d'année. Le but de ce projet consiste à implanter dans Cheddar de nouveaux algorithmes permettant de calculer, soit un temps de réponse des tâches, soit d'appliquer des tests de faisabilité bases sur le taux d'utilisation du processeur.

Ce rapport présente l'outil d'ordonnancement appelé Cheddar qui permet de simuler l'ordonnancement de tâches sur un ou plusieurs processeurs avec plusieurs protocoles d'ordonnancement et la conception de deux nouveaux algorithmes en ADA à l'aide d'un programme Rate Monotonic déjà existant, afin on les intègre dans Cheddar pour calculer les nouveaux algorithme, sous la direction de Monsieur Frank SINGHOFF. On choisit évidemment plusieurs jeux d'essai pour vérifier les nouveaux algorithmes sont correct.

Rappelons ce qu'est un test de faisabilité (ou test d'ordonnancabilité). Il s'agit d'un test qui indique par un calcul simple, si les échéances des tâches peuvent être respectées par une politique et ce sans avoir besoin d'exécuter l'algorithme.

Ce rapport est divisé en deux parties : La première partie présente l'ordonnancement temps réel et Cheddar avec notre environnement de travail. La deuxième partie présente les nouveaux algorithmes et les programme avec plusieurs jeux d'essai.

1.1 Système temps réel

Système temps réel lorsque ce système informatique contrôle (ou pilote) un procédé physique à une vitesse adaptée à l'évolution du procédé contrôlé. Les systèmes informatiques temps réel se différencient des autres systèmes informatiques par la prise en compte de contraintes temporelles dont le respect est aussi important que l'exactitude du résultat, autrement dit le système ne doit pas simplement délivrer des résultats exacts, il doit les délivrer dans des délais imposés.

Les systèmes temps réel sont présents dans de nombreux secteurs d'activités comme :

- L'industrie de production au travers des systèmes de contrôle de procédé.
- L'industrie du transport au travers des systèmes de pilotage embarque
- Les salles de marche au travers du traitement des données boursières en temps réel.
- La nouvelle économie au travers du besoin toujours croissant du traitement et l'acheminement de l'information.

L'ordonnancement consiste à organiser dans le temps la réalisation de tâches, compte tenu de contraintes temporelles (délais, contraintes d'enchainement) et de contraintes portant sur la disponibilité des ressources requises. Pour comprendre la méthode de calculer le temps de réponse dans le système temps réel, dans un premier temps on doit connaitre quelque définition

et les paramètres de formule.

1.1.1 Définitions

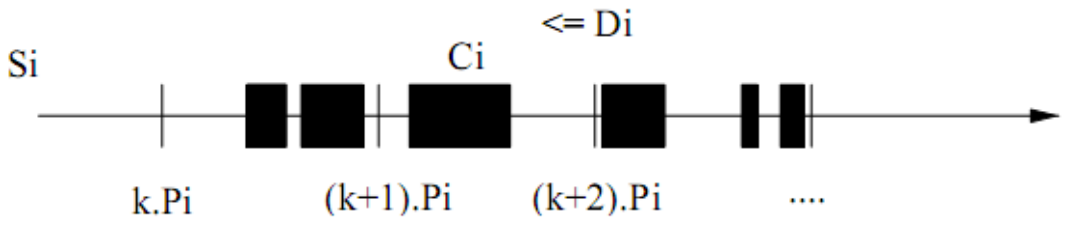
- Processeur : Unique ressource partageable dans un système temps réel → exécutif temps réel.
- Tâche : Suite d'instructions + données + contexte d'exécution (état).
- Famille de tâches :
  - Tâches dépendantes ou non. Tâches importantes, urgentes.
  - Tâches répétitives : activations successives (tâches périodiques ou sporadiques). → tâches critiques
  - Tâches non répétitives/apériodiques : une seule activation → tâches non critiques

Lorsqu'une tâche ne comporte qu'une occurrence, elle est dite apériodique. Si les occurrences d'une même tâche sont espacées par une constante Pi, la tâche sera dite périodique.

1.1.2 Paramètres

Une tâche est une suite séquentielle d'occurrence. Pour calculer le temps de réponse, on doit d'abord savoir les paramètres définissant une tâche i :

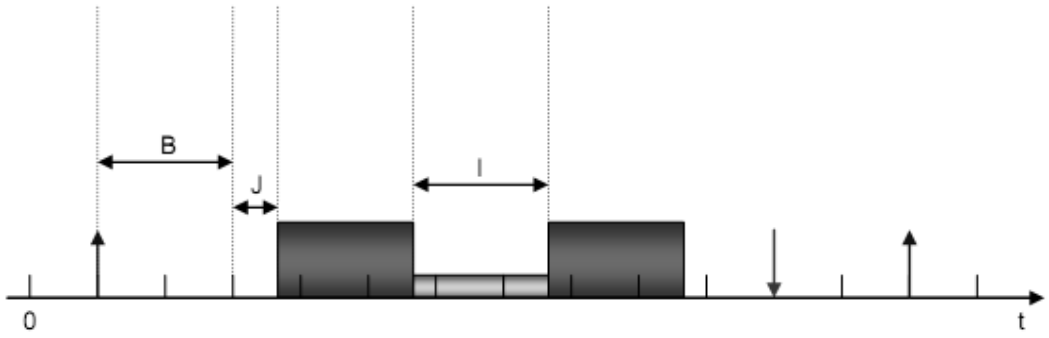
- La date de premier réveil (ou première activation) de la tâche dans le système, c'est-à-dire le premier instant où la tâche requiert le processeur. Le premier réveil de la tâche i est note Si.
- La durée d'exécution maximale de la tâche quand elle dispose du processeur pour elle seule. On parle aussi de capacité de la tâche, note Ci.
- Période d'activation : Pi. Pi est un délai fixe entre deux réveils successifs de la tâche i. pour chaque réveil, la tâche i doit exécuter un travail de Ci unités de temps.
- Délai critique : Di (relatif à Pi pour tâche périodique, à S_i pour tâche apériodique).
- La priorité de la tâche est une information qui permet à l'ordonnanceur de choisir la prochaine tâche prête à exécuter. Sa valeur est comprise entre 0 et 255.
- Date d'exécution au plus tôt : Ri.



Un modèle simplifie de tâches : le modèle de Liu et Lay land [LIU 73] ou modèle de tâches

périodiques à échéance sur requête. Caractéristiques :

- Tâches périodiques.
- Tâches indépendantes.
- Si = 0 → instant critique (pire cas).
- Pi = Di → tâches à échéances sur requêtes.



- Un temps d'instabilité au réveil (Jitter Time) qu'on peut aussi appeler gigue d'arrivée : J.
- Temps de blocage : B.
- le nombre d'invocations de la tâche i précédent à la m^{eme} invocation de la ti transaction, où ti = trans (i). : Q.
- L'interférence causée par une ou plusieurs autres tâches : I.

1.2 Travail à faire

- L'apprentissage d'ADA et l'outil Cheddar
- Apprendre le programme de calcul du temps de réponse Rate Monotonic.
- Modifier le calcul du temps de réponse Rate Monotonic pour deux nouveaux algorithmes dans le cas préemptif et le cas non préemptif.
- Intégrer les deux nouveaux programmes de calcul du temps de réponse dans l'outil Cheddar.

2. Environnement de travail

On travaille avec Cheddar pour développer les deux algorithmes. Il y a des versions Unix, Solaris et Windows qui sont disponible. Ici, nous choisissons la version Windows Cheddar pour créer les fichiers XML (jeu d'essai). Ensuite nous programmons les deux algorithmes sous Linux en langage ADA.

Le fait d'implémenter les formules dans Cheddar, puis on peut vérifier les résultats avec ceux obtenus par simulation, permet de montrer les erreurs dans l'implémentation des formules ou dans les formules elles même.

On peut aussi définir ses propres ordonnanceurs pour lesquels il n'existe pas de formules avec Cheddar. 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.

2.1 Cheddar

Cheddar est un outil de simulation permettant de calculer différents critères de performance (contraintes temporelles, dimensionnement de ressources). L'outil permet, entre autre, de tester le respect des contraintes temporelles d'un jeu de tâches modélisant une application/un système temps réel. Cet outil peut aider les personnes qui étudient/conçoivent des algorithmes d'ordonnancement temps réel ou qui souhaitent étudier les contraintes temporelles d'une application temps réel simple. Cheddar est maintenu par le LISyC, Université de Bretagne.

Cheddar est principalement constitué de deux composants logiciels :

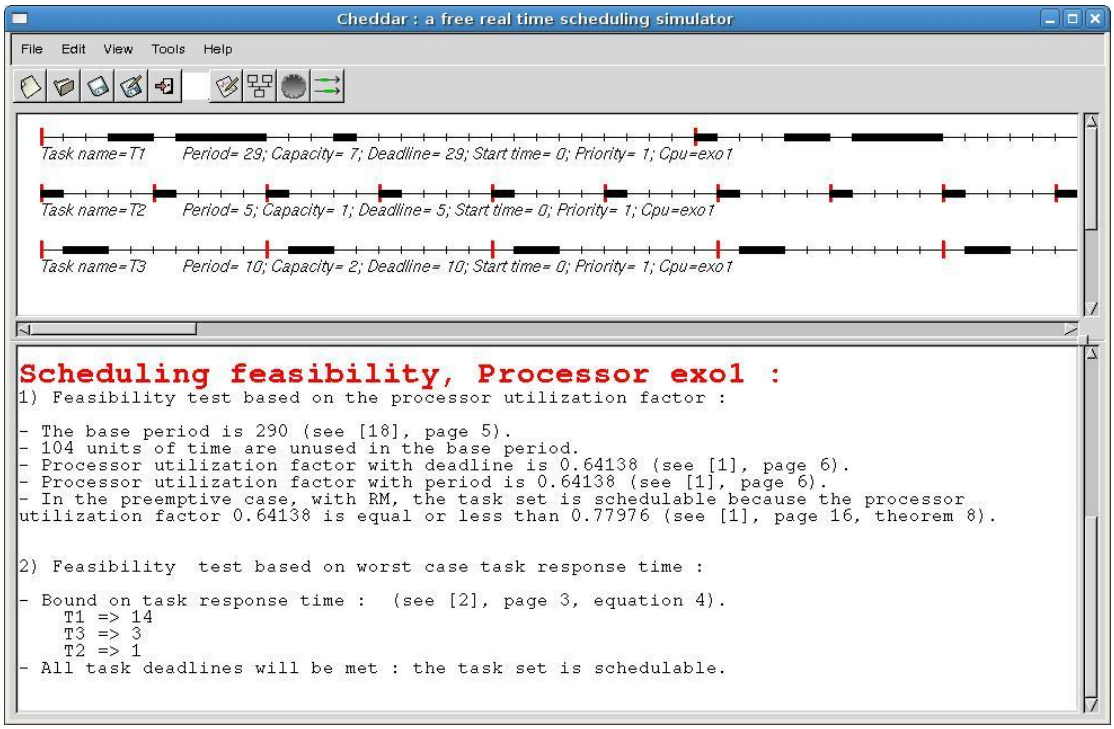
- Un éditeur qui permet à l'utilisateur de décrire l'application à analyser et sur lequel, les résultats de simulation seront présentés.
- Une bibliothèque comportant les principaux résultats de la théorie de l'ordonnancement temps réel ainsi que quelques outils de files d'attente.

Cheddar est écrit en Ada (l'interface graphique est basée sur GtkAda). L'outil a été testé sur Linux, Windows et Solaris mais devrait fonctionner sur toutes les plates-formes supportées par GNAT.

Cheddar offre deux types de fonctionnalité : un environnement de simulation et des tests de faisabilité. Les tests de faisabilité permettent à l'utilisateur de vérifier si les contraintes temporelles d'une application sont respectées sans pour autant calculer une simulation de l'ordonnancement. L'environnement de simulation quant à lui, effectue une analyse d'un ordonnancement précédemment calculé. L'environnement peut être étendu afin de modéliser des modèles de tâches ou d'ordonnanceurs non couverts par la théorie de l'ordonnancement actuel.

La figure suivante présente l'interface graphique principale avec un résultat de simulation. Dans

la fenêtre du haut, l'outil présente le chronogramme de la simulation et pour chaque résultat d'analyse, Cheddar affiche le nom ou la référence de la méthode de calcul appliquée. Dans celle du bas, l'outil présente les informations sur le projet en cours comme les tests de faisabilité hors ligne, ou encore les statistiques de la simulation



2.2 Langage ADA

Ada est un langage de programmation conçu par l'équipe de CII-Honeywell Bull dirigée par Jean Ichbiah en réponse à un cahier des charges établi par le département de la Défense des États-Unis (DoD).

Ada est souvent utilisée dans des systèmes temps réel et embarqués nécessitant un haut niveau de fiabilité et de sécurité. De nos jours, Ada95 est employé bien sûr par son initiateur, mais aussi dans toutes les technologies de pointes. Il est possible de trouver des compilateurs Ada pour certains systèmes d'exploitation (Windows, Linux, VxWorks) et d'architectures matérielles, y compris un compilateur libre (GNAT, inclus dans GNU Compiler Collection) compilant de l'Ada 83/95/2005.

Il est souvent utilisé en introduction aux cours de programmation informatique avancée, et parce qu'il partage les mêmes qualités pédagogiques que le Pascal (dont il a hérité), Ada est même maintenant utilisé pour les cours d'introduction à la programmation.

Le compilateur Ada le plus utilise est Gnat. GNAT est le compilateur Ada du projet GNU. Il fait partie de GNU Compiler Collection (GCC). Une table des symboles (les fichiers .ali) est créée pour

chaque package afin d'accélérer la recherche des symboles dans les fichiers et donc détecter plus vite les erreurs de compilation. Gnat est livre avec une suite d'exécutables, donc le plus intéressant est gnatmake, il compile un fichier cible et toutes ses dépendances. Il n'est pas forcement de s'affranchir de l'écriture des Makefiles.

2.3 Système

- Windows XP SP3
- Ubuntu Linux 8.10
- GCC 4.3
- GNAT 4.3

3. Algorithme à programmer

Pour la partie de programme, dans un premier temps on étudie un programme qui calcule l'algorithme Rate Monotonic. Nous avons déjà étudié cet algorithme en Master 1 dans le cours Système temps réel. En suite il y a deux algorithmes à programmer à base de premier programme. Un algorithme préemptif et un algorithme non préemptif.

3.1 Rate Monotonic

Rate Monotonic est un algorithme à priorité fixe, utilisé uniquement pour les tâches périodiques. La tâche qui a la priorité la plus forte, est élue pour être exécutée. La priorité est l'inverse de la périodicité : la tâche ayant la période la plus petite, dans un jeu de tâches, est la plus prioritaire. C'est un algorithme optimal dans la classe des algorithmes à priorité fixe, facile à implanter dans un système d'exploitation. Les tâches sont ordonnancables en mode préemptif ou non préemptif.

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{P_j} \right\rceil C_j$$

- On démarre avec $W_i^0=C_i$.
- Echec si $W_i^n>P_i$.
- Réussite si $W_i^{n+1}=W_i^n$.

3.2 Préemptif

3.2.1. Méthode de calcul :

$$r_i = \max_{q=0,1,2,\ldots} (J_i + w_i(q) - qP_i)$$

Avec

$$w_i(q) = (q+1)C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{J_j + w_i(q)}{P_j} \right\rceil C_j$$

Tel que :

$$\forall q : w_i(q) \geq (q+1).P_i$$

3.2.2. Références : [JP86, ABRT93, TC94].

3.2.3. Hypothèses :

- Algorithme préemptif à priorité fixe, quelque soit l'algorithme d'affectation des priorités.
- Tâches à échéances quelconques, concrètes et synchrones.
- Tâches dépendantes (ressources partagées et/ou contraintes de précédence exprimée avec une gigue).

3.2.4. Commentaire :

- hp(i) est l'ensemble des tâches de plus forte priorité que i.
- Condition nécessaire et suffisante si la gigue et le temps de blocage sur ressources partagées ne sont pas employés. Condition suffisante seulement dans le cas contraire.

3.2.5 Code source

```
--source ~singhoff/PUBLIC/ada.csh
--cd CHEDDAR-2.0/trunk/framework_examples
--make alone "TARGET=compute_response_time"
--./compute_response_time

with Scheduler;                    use Scheduler;
with Tasks;                        use Tasks;
with Task_Set;                     use Task_Set;
use Task_Set.Generic_Task_Set;
with Ada.Strings.Unbounded;        use Ada.Strings.Unbounded;
with Ada.Numerics.Aux;             use Ada.Numerics.Aux;
with Time_Units;                   use Time_Units;
use Time_Units.A_Time_Unit;
with Task_Dependencies;            use Task_Dependencies;
use Task_Dependencies.Half_Dep_Set;
with Call_Framework;               use Call_Framework;
with Systems;                      use Systems;
with Unbounded_Strings;            use Unbounded_Strings;
use Unbounded_Strings.Unbounded_String_List_Package;
with double_util;                  use double_util;
with framework_config;             use framework_config;
with text_io;                      use text_io;
use Scheduler.Double_Tasks_Parameters;


procedure compute_response_time is
```

```
function Compute_Workload (
        My_Tasks        : in        Tasks_Set;              --all the task
        Current_Task : in        Generic_Task_Ptr;     --the task examine
        q                    : in        Integer
        )return Double is
    Iterator2 : tasks_Iterator;
    Taskj        : Generic_Task_Ptr;
    Wiq_k,
    Wiq_k1       : Double;
begin
    Wiq_k:=-0.1;
    Wiq_k1:=0.0;

    -- approximations for Wiq
    -- exit loop when approximations converge
    while Wiq_k1 /= Wiq_k    loop
        Reset_Iterator(My_Tasks, Iterator2);
        Wiq_k:=Wiq_k1;
        Wiq_k1:=Double((q+1)*Current_Task.Capacity
                    + Current_Task.Blocking_Time);

        -- Iterate on all tasks
        loop
            Current_Element(My_Tasks, Taskj, Iterator2);

            -- If Taskj is in hp(Taski)
            if (Taskj.Priority>Current_Task.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 loop when there is no more task
            exit when Is_Last_Element(My_Tasks, Iterator2);
            Next_Element(My_Tasks, Iterator2);
        end loop;
    end loop;
    return Wiq_k1;
end Compute_Workload;
```

```
        Response_Time    :        Response_Time_Table;
        Tmp              : Tasks_Set;
        Iterator1 : tasks_Iterator;
        Taski,
        Task_Tmp    : Generic_Task_Ptr;
        Wiq,
        Ri               : Double                    := 0.0;
        dir              : Unbounded_String_List;
        Sys              : System;
        Q                : Integer;
        I                : Response_Time_Range := 0;
    Begin
        Call_Framework.initialize (False);
        Initialize (Sys);

        --read the data and save in the Sys
        --dir: for various dossier
        --change the rm4_net.xml to your file xml.
        Read_From_Xml_File (Sys, dir, To_Unbounded_String ("rm4_net.xml"));
        Reset_Iterator(sys.Tasks, Iterator1);

        --to sort by priority
        loop
            Current_Element(sys.Tasks, Taski, Iterator1);
            if (Taski.cpu_name = To_Unbounded_String("cpu"))
            then
                Task_Tmp:= new Periodic_Task;
                Task_Tmp.all:=Taski.all;
                Add(Tmp,Task_Tmp);
            end if;
            exit when Is_Last_Element(sys.Tasks, Iterator1);
            Next_Element(sys.Tasks, Iterator1);
        end loop;

        --Now, compute response time
        Sort(Tmp, Increasing_Priority'access);
        Reset_Iterator(Tmp, Iterator1);
        i:=0;
        loop

            --select the 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;

            -- Find the maximum for Ri in [0..Q]
            Q := 0;
            loop
                Wiq := Compute_Workload (Tmp, Taski, Q);
                case Taski.Task_Type is
                    when Periodic_Type =>
                        Ri := Wiq +
                                Double (Periodic_Task_Ptr (Taski).Jitter) -
                                Double (Q) *
                                Double (Periodic_Task_Ptr (Taski).Period);
                    when others =>
                        raise Constraint_Error;
                end case;

                -- select the maximum for Ri ?
                if (Ri > Response_Time.Entries (I).Data)
                then
                    Response_Time.Entries (I).Data := Ri;
                end if;
                exit when Wiq<=Double(Q+1)*Double(Periodic_Task_Ptr(Taski).Period);
                Q:=Q+1;
            end loop;
            Put_Line("Task "
                    & To_String(Response_Time.Entries (I).Item.Name)
                    & " = "
                    & To_String(Format(Response_Time.Entries (I).Data)) );
            exit when Is_Last_Element(Tmp, Iterator1);
            Next_Element(Tmp, Iterator1);
            I:=I+1;
        end loop;
end compute_response_time;
```

3.3 Non préemptif

3.3.1 Méthode de calcul :

$$r_i = max_{q = 0, \ldots, Q}\{w_{i,q} + C_i - qT_i\},$$

$$w_{i,q} = qC_i + \sum_{j \in hp(i)} \left(1 + \left\lfloor \frac{w_{i,q}}{T_j} \right\rfloor\right) C_j + max_{k \in lp(i)}\{C_k - 1\},$$

Avec

$$Q = \lfloor L_i / T_i \rfloor,$$

Un level-i busy period est un Période occupée de Processeur, dans lequel les instances des tâches avec une priorité supérieure ou égale à celle de τi exécuter.

Pour calculer le pire cas temps de réponse, il est nécessaire de calculer la longueur d'aptitude level-i busy periods. L'approche est très similaire à celle pour le calcul de la synchrone durée de la période occupée. Dans ce cas, seul la charge de travail des tâches avec une priorité supérieure ou égale à celle de τi est pris en compte.

La longueur de la plus longue de level-i période occupé est désignée par Li, et peut facilement être calculée par la recherche la plus petite solution de l'équation

Pour le cas préemptif :

$$L_i = \sum_{j \in hp(i) \cup \{i\}} \left\lceil \frac{L_i}{T_j} \right\rceil C_j,$$

Pour les cas non-préemptif,

$$L_i = max_{j \in lp(i)}\{C_j - 1\} + \sum_{j \in hp(i) \cup \{i\}} \left\lceil \frac{L_i}{T_j} \right\rceil C_j,$$

Notez que dans ce dernier cas, Comme d'habitude, les équations peuvent être résolues par des moyens de calculs itératifs.

3.3.2 Code source

```
--source ~singhoff/PUBLIC/ada.csh
--cd CHEDDAR-2.0/trunk/framework_examples
--make alone "TARGET=compute_response_time_nonpreemtive"
--./compute_response_time_nonpreemtive
```

```
with Scheduler;                          use Scheduler;
with Tasks;                              use Tasks;
with Task_Set;                           use Task_Set;
use Task_Set.Generic_Task_Set;
with Ada.Strings.Unbounded;              use Ada.Strings.Unbounded;
with Ada.Numerics.Aux;                   use Ada.Numerics.Aux;
with Time_Units;                         use Time_Units;
use Time_Units.A_Time_Unit;
with Task_Dependencies;                  use Task_Dependencies;
use Task_Dependencies.Half_Dep_Set;
with Call_Framework;                     use Call_Framework;
with Systems;                            use Systems;
with Unbounded_Strings;                  use Unbounded_Strings;
use Unbounded_Strings.Unbounded_String_List_Package;
with double_util;                        use double_util;
with integer_util;                       use integer_util;
with framework_config;                   use framework_config;
with text_io;                            use text_io;
use Scheduler.Double_Tasks_Parameters;

procedure compute_response_time_nonpreemtive is

    function Compute_Workload (
            My_Tasks      : in      Tasks_Set;          --all the task
            Current_Task : in      Generic_Task_Ptr; --the task examine
            q             : in      Integer
            )return Double is
        Iterator2 : tasks_Iterator;
        Iterator3 : tasks_Iterator;
        Taskk,
        Taskj      : Generic_Task_Ptr;
        Ck,
        Ck_tmp,
        Wiq_k,
        Wiq_k1     : Double;
    begin
        Wiq_k:=-0.1;
        Wiq_k1:=1.0;
        Ck:=0.0;
        Ck_tmp:=0.0;

        -- approximations for Wiq
        -- exit loop when approximations converge
        while Wiq_k1 /= Wiq_k    loop
```

```
            Reset_Iterator(My_Tasks, Iterator2);
            Wiq_k:=Wiq_k1;
            Wiq_k1:=Double(q*Current_Task.Capacity);

            -- Iterate on all tasks
            loop
                Current_Element(My_Tasks, Taskj, Iterator2);

                -- If Taskj is in hp(Taski)
                if (Taskj.Priority>Current_Task.Priority)
                then
                    Wiq_k1:=Wiq_k1
                        + (
                            Double'Floor( Wiq_k / Double(Periodic_Task_Ptr(Taskj).Period))
                            + 1.0
                          )
                        * Double(Taskj.Capacity);
                end if;

                -- exit loop when there is no more task
                exit when Is_Last_Element(My_Tasks, Iterator2);
                Next_Element(My_Tasks, Iterator2);
            end loop;

            --calcul the max{Ck-1}
            loop
                Current_Element(My_Tasks, Taskk, Iterator3);
                if (Taskk.Priority<Current_Task.Priority)
                then
                    Ck_tmp := Double(Taskk.Capacity)-1.0;
                    if ( Ck_tmp >Ck )
                    then
                        Ck := Ck_tmp;
                    end if;
                end if;
                exit when Is_Last_Element(My_Tasks, Iterator3);
                Next_Element(My_Tasks, Iterator3);
            end loop;
            Wiq_k1:=Wiq_k1+Ck;
        end loop;
        return Wiq_k1;
    end Compute_Workload;

    --calcul the value of level-i busy period
```

```
function compute_L (
        My_Tasks        : in        Tasks_Set;
        Current_Task : in        Generic_Task_Ptr
        )return Double is
    Iterator2 : tasks_Iterator;
    Taskj2      : Generic_Task_Ptr;
    Ln,
    Ln1       : Double;
begin
    Ln:=-0.1;
    Ln1:=1.0;
    while Ln1 /= Ln    loop
        Reset_Iterator(My_Tasks, Iterator2);
        Ln:=Ln1;
        Ln1:=0.0;
        loop
            Current_Element(My_Tasks, Taskj2, Iterator2);
            if (Taskj2.Priority>=Current_Task.Priority)
            then
                Ln1:=Ln1
                    + Double'Ceiling(
                            Ln / Double(Periodic_Task_Ptr(Taskj2).Period)
                       )
                       * Double(Taskj2.Capacity);
            end if;
            exit when Is_Last_Element(My_Tasks, Iterator2);
            Next_Element(My_Tasks, Iterator2);
        end loop;
    end loop;
    return Ln1;
end compute_L;
    Response_Time    :        Response_Time_Table;
    Tmp            : Tasks_Set;
    Iterator1 : tasks_Iterator;
    Taski,
    Task_Tmp    : Generic_Task_Ptr;
    Li,
    Q_double,
    Wiq,
    Ri             : Double                    := 0.0;
    dir            : Unbounded_String_List;
    Sys            : System;
    Q_arrete,
    Q              : Integer;
```

```
        I               : Response_Time_Range := 0;
    begin

        Call_Framework.initialize (False);
        Initialize (Sys);

        --read the data and save in the Sys
        --dir: for various dossier
        --change the test_RT_non_preemptive3.xml to your file xml.
        Read_From_Xml_File (Sys, dir, To_Unbounded_String ("test_RT_non_preemptive3.xml"));
        Reset_Iterator(sys.Tasks, Iterator1);

        --to sort by priority
        loop
            Current_Element(sys.Tasks, Taski, Iterator1);
            if (Taski.cpu_name = To_Unbounded_String("cpu"))
            then
                Task_Tmp:= new Periodic_Task;
                Task_Tmp.all:=Taski.all;
                Add(Tmp,Task_Tmp);
            end if;
            exit when Is_Last_Element(sys.Tasks, Iterator1);
            Next_Element(sys.Tasks, Iterator1);
        end loop;
        Sort(Tmp, Increasing_Priority'access);
        Reset_Iterator(Tmp, Iterator1);
        I:=0;
        loop

            --select the task
            Current_Element(Tmp, Taski, Iterator1);

            --initialiser le temps de reponse pr 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;

            -- Find the maximum for Ri in [0..Q]
            Q := 0;
            Li := Compute_L (Tmp, Taski);
            loop
                Q_double := Double'Floor(Li / Double(Periodic_Task_Ptr(Taski).Period));
                Q_arrete := Integer(Q_double);
                Wiq := Compute_Workload (Tmp, Taski, Q);
```

```
                --Ri = Wiq + Ci - qTi
                Ri := Wiq +
                        Double (Periodic_Task_Ptr (Taski).Capacity) -
                        Double (Q) * Double (Periodic_Task_Ptr (Taski).Period);

                -- Select maximum for Ri ?
                if (Ri > Response_Time.Entries (I).Data)
                then
                    Response_Time.Entries (I).Data := Ri;
                end if;

                exit when Q = Q_arrete;
                Q:=Q+1;
            end loop;
            Put_Line("Task "
                    & To_String(Response_Time.Entries (I).Item.Name)
                    & " = "
                    & To_String(Format(Response_Time.Entries (I).Data)) );
            exit when Is_Last_Element(Tmp, Iterator1);
            Next_Element(Tmp, Iterator1);
            I:=I+1;
        end loop;
end compute_response_time_nonpreemtive;
```

4. Jeu d'essai

Pour vérifier notre programme des algorithmes est juste, nous avons choisi plusieurs jeux de tests à tester notre programme pour chaque algorithme.

4.1 Listes des tests et le temps de réponse (préemptive)

4.1.1 test_RT_preemptive1.xml (Période = Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	7	3	7	3	***3***
T2	12	2	12	2	***5***
T3	20	5	20	1	***18***

4.1.2 test_RT_preemptive2.xml (Période = Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	29	7	29	1	***14***
T2	5	1	5	3	***1***
T3	10	2	10	2	***3***

4.1.3 test_RT_preemptive3.xml (Période = Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	30	6	30	1	***30***
T2	5	3	5	3	***3***
T3	10	2	10	2	***5***

4.1.4 test_RT_preemptive4.xml (Période = Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	50	25	50	2	***25***
T2	100	40	100	1	***90***

4.1.5 test_RT_preemptive5.xml (Période < Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	7	3	10	3	***3***
T2	12	2	20	2	***5***
T3	20	5	40	1	***18***

4.1.6 test_RT_preemptive6.xml (Période < Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	29	7	32	1	***14***
T2	5	1	10	3	***1***
T3	10	2	10	2	***3***

4.1.7 test_RT_preemptive7.xml (Période < Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	30	6	40	1	***30***
T2	5	3	10	3	***3***
T3	10	2	20	2	***5***

4.1.8 test_RT_preemptive8.xml (Période < Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	50	25	60	2	***25***
T2	100	40	150	1	***90***

4.1.9 test_RT_preemptive9.xml (Période > Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	7	3	5	3	***3***
T2	12	2	8	2	***5***
T3	20	5	18	1	***18***

4.1.9 test_RT_preemptive9_miss.xml (Période > Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	7	3	5	3	***3***
T2	12	2	9	2	***5***
T3	20	5	16	1	***18(miss)***

4.1.10 test_RT_preemptive10.xml (Période > Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	29	7	15	1	***14***
T2	5	1	4	3	***1***
T3	10	2	8	2	***3***

4.1.10 test_RT_preemptive10_miss.xml (Période > Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	29	7	8	1	***14(miss)***
T2	5	1	4	3	***1***
T3	10	2	8	2	***3***

4.1.11 test_RT_preemptive11_miss.xml (Période > Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	30	6	29	1	***30(miss)***
T2	5	3	4	3	***3***
T3	10	2	6	2	***5***

4.1.12 test_RT_preemptive12.xml (Période > Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	50	25	40	2	***25***
T2	100	40	90	1	***90***

4.1.12 test_RT_preemptive12_miss.xml (Période > Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	50	25	40	2	***25***
T2	100	40	50	1	***90(miss)***

4.2 Listes des tests et le temps de réponse (non préemptive)

4.2.1 test_RT_non_preemptive1.xml (Période = Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	7	3	7	3	***7***
T2	12	2	12	2	***12***
T3	20	5	20	1	***10***

4.2.2 test_RT_non_preemptive2.xml (Période = Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	29	7	29	1	***10***
T2	5	1	5	3	***7(miss)***
T3	10	2	10	2	***10***

4.2.3 test_RT_non_preemptive3.xml (Période = Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	30	6	30	1	***14***
T2	5	3	5	3	***8(miss)***
T3	10	2	10	2	***16(miss)***

4.2.4 test_RT_non_preemptive4.xml (Période = Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	50	25	50	2	***64(miss)***
T2	100	40	100	1	***65***

4.2.5 test_RT_non_preemptive5.xml (Période < Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	7	3	10	3	***7***
T2	12	2	20	2	***12***
T3	20	5	40	1	***10***

4.2.6 test_RT_non_preemptive6.xml (Période < Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	29	7	32	1	***10***
T2	5	1	10	3	***7***
T3	10	2	10	2	***10***

4.2.7 test_RT_non_preemptive7.xml (Période < Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	30	6	40	1	***14***
T2	5	3	10	3	***8***
T3	10	2	20	2	***16***

4.2.8 test_RT_non_preemptive8.xml (Période < Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	50	25	60	2	***64(miss)***
T2	100	40	150	1	***65***

4.2.9 test_RT_non_preemptive9.xml (Période > Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	7	3	5	3	***7(miss)***
T2	12	2	8	2	***12(miss)***
T3	20	5	18	1	***10***

4.2.9 test_RT_non_preemptive9_miss.xml (Période > Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	7	3	5	3	***7(miss)***
T2	12	2	9	2	***12(miss)***
T3	20	5	16	1	***10***

4.2.10 test_RT_non_preemptive10.xml (Période > Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	29	7	15	1	***10***
T2	5	1	4	3	***7(miss)***
T3	10	2	8	2	***10(miss)***

4.2.10 test_RT_non_preemptive10_miss.xml (Période > Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	29	7	8	1	***10(miss)***
T2	5	1	4	3	***7(miss)***
T3	10	2	8	2	***10(miss)***

4.2.11 test_RT_non_preemptive11_miss.xml (Période > Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	30	6	29	1	***14***

T2	5	3	4	3	***8(miss)***
T3	10	2	6	2	***16(miss)***

4.2.12 test_RT_non_preemptive12.xml (Période > Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	50	25	40	2	***64(miss)***
T2	100	40	90	1	***65***

4.2.12 test_RT_non_preemptive12_miss.xml (Période > Deadline)

Tâche	Période	Capacité	Deadline	Priorité	***Ri***
T1	50	25	40	2	***64(miss)***
T2	100	40	50	1	***65(miss)***

5. Bilan et Conclusion

Ce projet m'a permis d'approfondir mes connaissances des ordonnancements temps réels. 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.

A cause du problème de la durée de projet, on a fini deux programmes : un algorithme préemptif à priorité fixe et un algorithme non préemptif.

Pour conclure, il est important de souligner qu'il existe encore des travaux à effectuer dans ce domaine. En effet, si on a encore le temps, j'aimerais bien continuer programmer un autre nouvel algorithme qui est publie dans une publication <ADDING TIME-OFFSETS TO SCHEDULABILITY ANALYSIS> par Ken Tindell en page 8 :

$$r_i = \max_{q=0,1,2,3,\cdots} \left(\max_{\forall j \in tasks(ti)} \left(w_{i,q} + O_j + J_j - T_{ti}\left(qe_i + v_{i,ti}\right) - O_i \right) \right)$$

Avec

$$w_{i,q} = B_i + (q+1)C_i + \sum_{\forall t \in tasks} I_{i,t}$$

Et

$$v_{i,ti} = \left\lceil \frac{O_j + J_j - O_i - J_i}{T_{ti}} \right\rceil$$

Dans cet algorithme, il y a une addition time offset à la tâche priorité fixe

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

Références

[1] Frank SINGHOFF - <A propos de l'applicabilité de la théorie de l'ordonnancement temps réel : le projet Cheddar>

[2] laurent George, Nicolas Rivierre, Marco Spuri - <Preemptive and Non-preemptive Real-Time Uni-Processor Scheduling>

[3] John Barnes - <Ada 2005 Rationale>

[4] Frank SINGHOFF - <Ordonnancement temps réel monoprocesseur>