

TP4 ordonnancement temps réel

Table des matières

- [Introduction](#)
- [I. Prise en main de l'outil Cheddar](#)
 - [Configuration de l'environnement de travail](#)
 - [Elaboration du modèle à analyser](#)
 - [Charger/sauvegarder un modèle](#)
 - [Analyse d'un modèle](#)
 - [Exercice 1](#)
- [II. Comparaisons d'algorithmes](#)
 - [Exercice 2](#)
 - [Exercice 3](#)
- [III. Partage de ressources](#)
 - [Exercice 4](#)
 - [Exercice 5](#)
- [IV. Exercices de synthèse](#)
 - [Exercice 6](#)
 - [Exercice 7](#)
- [V. Références](#)

Introduction

L'objectif de ce TP est d'expérimenter quelques algorithmes et méthodes classiques pour l'analyse d'ordonnancement de systèmes temps réel. Pour ce faire, nous allons utiliser un outil qui implante plusieurs de ces méthodes : l'outil Cheddar. Le premier exercice est essentiellement un exemple de prise en main de l'outil. Dans ce premier exercice, nous regardons comment éditer un modèle d'un système temps réel avec Cheddar, puis comment en réaliser son analyse. Par la suite :

1. Les exercices 2 et 3 présentent et comparent divers algorithmes d'ordonnancement classiques.
2. Les exercices 4 et 5 abordent le partage de ressources entre tâches.
3. Enfin les exercices 6 et 7 proposent des exercices de synthèse.

I. Prise en main de l'outil Cheddar

I.1 Configuration de l'environnement de travail

- Cheddar a été installé sur les PC utilisés pour ce tutoriel. Vous pouvez ouvrir le dossier Cheddar trouvé dans `~/bin`
 - Les autres distributions de Cheddar sont également disponibles en téléchargement [ici](#)
- Dans un terminal, exécuter `./cheddar`
- La fenêtre de la figure 1 doit s'afficher :

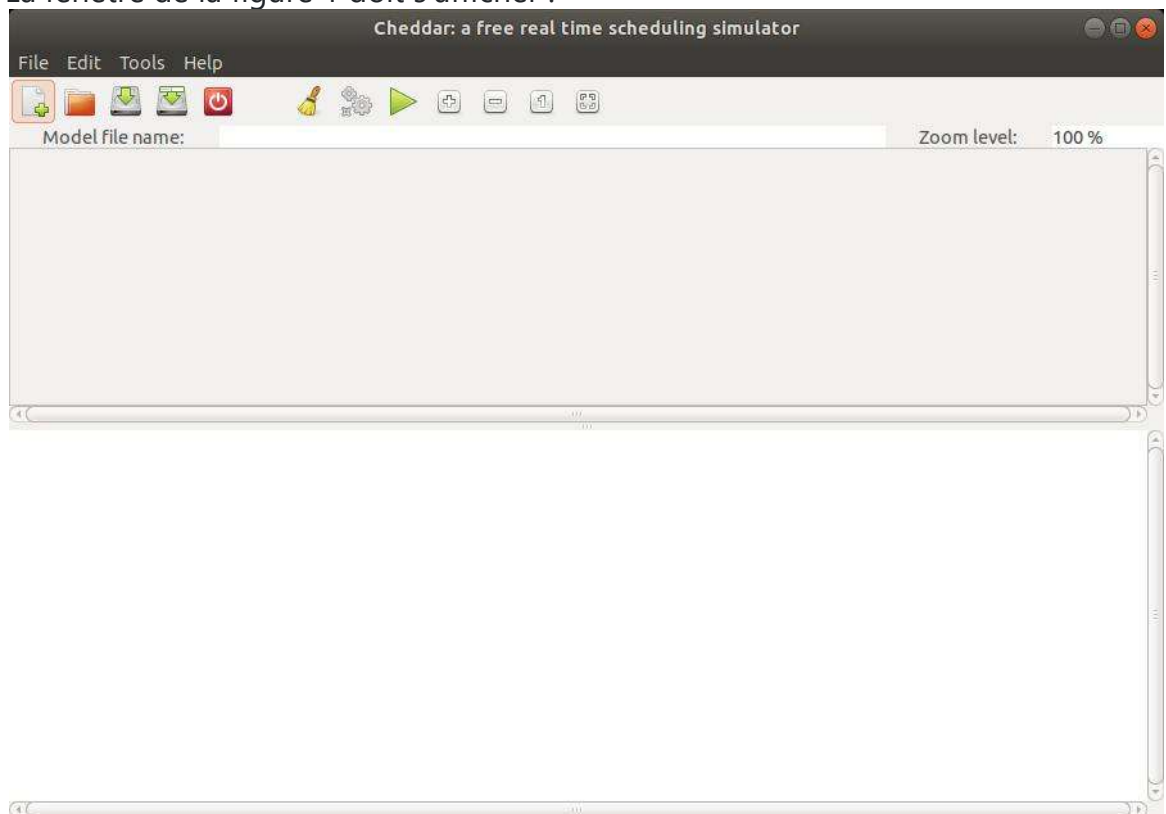


Fig. 1: Fenêtre principale de Cheddar

- La partie haute (en gris pour l'instant) de cette fenêtre affiche des chronogrammes représentant la simulation du système temps réel étudié. La partie basse (en blanc) de cette fenêtre affiche divers résultats d'analyse.

I.2 Elaboration du modèle à analyser

Dans cette partie, vous allez construire votre premier modèle Cheddar en suivant les consignes ci-dessous.

Pour effectuer une analyse d'ordonnancement, il est dans un premier temps nécessaire de décrire le système à analyser avec Cheddar ADL. Cheddar ADL est un langage d'architecture simplifié et dédié à l'analyse d'ordonnancement. Il comporte une description de la partie matérielle et de la partie logicielle du système à analyser. Ces modèles peuvent être rédigés avec n'importe quel éditeur de texte, toutefois, par

la suite, nos modèles seront élaborés avec l'éditeur interne de Cheddar de la façon suivante :

1. Dans un premier temps, la plate-forme d'exécution matérielle est décrite. Pour Cheddar, cette plate-forme peut être constituée d'un ou plusieurs composants **processor** incluant un ou plusieurs composants **core** et éventuellement des composants **cache**. Un **core** est ici une unité de calcul, i.e. une entité qui offre des moyens pour exécuter une ou plusieurs tâches.
2. Pour décrire un processeur comportant un **core**, utilisez le menu **Edit/Hardware/Core** qui permet d'ouvrir la fenêtre de la figure 2 :

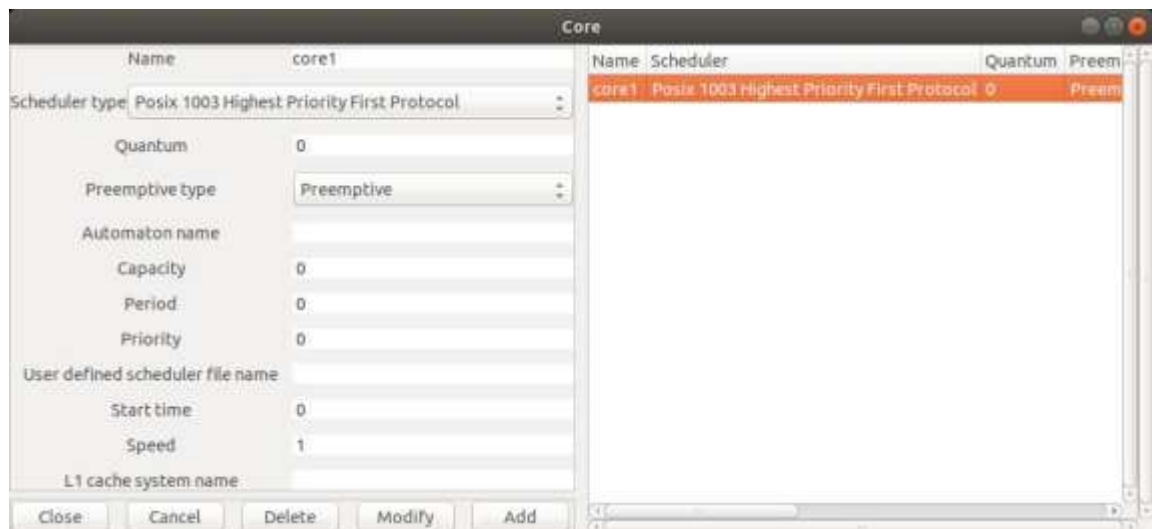


Fig. 2 : Édition d'un composant core

3. Un composant **core** est défini par les attributs suivants:
 - **Name** : le nom unique du composant, ici, **core1**.
 - **Scheduler type** : la méthode d'ordonnancement qui sera appliquée par le composant. Dans la fenêtre ci-dessus, nous avons défini un **core** hébergeant un ordonnanceur à priorité fixe conforme au standard POSIX 1003 (valeur **POSIX_1003_Highest_Priority_First_Protocol** pour l'attribut **Scheduler type**). Dans Cheddar, avec cet ordonnanceur, les priorités fixes peuvent varier de 0 à 255 (255 étant la priorité la plus forte). Les priorités de 1 à 255 sont réservées aux politiques POSIX SCHED_RR et SCHED_FIFO. Le niveau de priorité 0 est réservé à la politique SCHED_OTHER uniquement. Ces politiques POSIX permettent de déterminer l'ordonnancement d'un ensemble de tâches dont la priorité fixe est identique. Notez que cette configuration de priorité et de politique est celle implantée dans Linux.
 - **Preemptive type** : permet de spécifier si l'ordonnanceur doit être préemptif ou non.
 - Les autres champs peuvent être ignorés à ce stade.

4. Vous trouverez ci-dessous un bref descriptif des différents boutons de cette fenêtre, boutons qui seront identiques dans toutes les fenêtres suivantes :
 - Bouton **Add** : il permet d'ajouter un nouveau composant.
 - Bouton **Delete** : il permet de supprimer un composant existant après l'avoir sélectionné en cliquant sur la liste des composants dans la partie droite de la fenêtre.
 - Bouton **Modify** : il permet de modifier un composant existant après l'avoir sélectionné en cliquant sur la liste des composants dans la partie droite de la fenêtre et après avoir modifié la valeur de ses attributs.
 - Bouton **Close** : il permet de valider l'ensemble des modifications effectuées depuis l'ouverture de la fenêtre.
 - Bouton **Cancel** : il permet d'annuler toutes les modifications effectuées depuis l'ouverture de la fenêtre.
5. Une fois **core1** défini, il est possible de décrire le **processor** qui utilisera **core1** avec le menu **Edit/Hardware/Processor**, qui affiche la fenêtre ci-dessous:

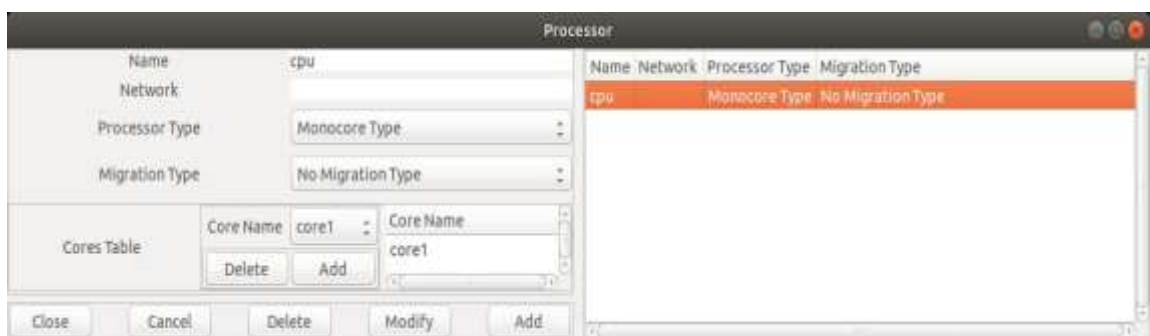


Fig. 3 : Édition d'un composant processor

Dans la figure 3, nous définissons un processeur **cpu1** comportant uniquement le core **core1**. L'ajout et la suppression d'un core pour un processeur s'effectue par les boutons **Add** et **Delete** dans la partie **Cores Table**. Puisque nous nous limitons ici à un **processor** comportant un seul **core**, Cheddar requiert alors les valeurs **Monocore Type** et **No Migration Type** pour les attributs respectifs **Processor Type** et **Migration Type**.

6. Par la suite, il est nécessaire de décrire la partie logicielle du système à analyser : il s'agit ici des tâches (composant **task**) mais aussi des ressources partagées entre les tâches (composant **resource**). Cheddar requiert par ailleurs la modélisation des entités mémoires regroupant tâches et ressources, c'est le rôle des composants **Address Space**.
7. La fenêtre de la figure 4 (menu **Edit/Software/Address space**) montre la définition d'un composant **Address Space** pour notre exemple. Seuls les champs **Name** et **Cpu Name** doivent être renseignés ici. Lors des exercices de ce tutoriel, nous devons définir un composant **Address Space** pour chaque

processeur. La plupart des attributs de ce composant est utilisée pour l'analyse d'ordonnancements hiérarchiques comme ceux rencontrés dans les systèmes ARINC 653. Ainsi, les autres champs de cette fenêtre n'ont pas besoin d'être renseignés pour ce tutoriel.

Fig. 4 : Édition d'un composant address space

8. La dernière étape consiste en la description des tâches du système avec le menu **Edit/Software/Task** :

Fig. 5: édition d'un composant task

Une tâche peut être définie par de nombreux attributs. Les plus importants sont :

- **Name** : le nom (unique) du composant.
- **Task type** : le type de tâche qui spécifie les instants de réveil de la tâche (ex : périodique, sporadique, aléatoire selon un processus de poisson, ...).
- **Capacity** : le WCET de la tâche.
- **Period** : le délai entre deux réveils successifs de la tâche, notamment la période lorsque la tâche est périodique.
- **Deadline** : l'échéance à respecter par la tâche. Il s'agit ici d'une **échéance relative** à la date de réveil de la tâche.
- **Address space name** et **CPU name** qui indique sur quels composants **Address space** et **Processor** la tâche est placée.
- **Priority** : la priorité fixe associée à la tâche.

Dans l'exemple de la figure 5, nous avons défini 3 tâches, périodiques, synchrones et à échéances sur requêtes, avec les paramètres suivants:

- Tâche T1 : **Capacity** = 2, **Deadline** = 6, **Period** = 6
- Tâche T2 : **Capacity** = 2, **Deadline** = 9, **Period** = 9
- Tâche T3 : **Capacity** = 3, **Deadline** = 12, **Period** = 12
- Les priorités (attribut **Priority**) ont été affectées selon Rate Monotonic dans la plage de priorité autorisée (de 1 à 255).
- Notez que les délais critiques sont égaux aux périodes : la valeur de l'attribut **Deadline** est égale à celle de l'attribut **Period**.
- Enfin, les tâches sont synchrones ; l'attribut **Start time** est égal à 0 pour toutes les tâches.

I.3 Charger/Enregistrer un modèle

Cheddar fournit des fonctionnalités pour charger/enregistrer un modèle de système au format Cheddar ADL.



- **File/Save XML Project** : un système modélisé en Cheddar peut être exporté dans un fichier XML respectant le format Cheddar ADL.
- **File/Open XML Project**: un modèle peut être chargé dans Cheddar pour être analysé. Dans ce tutoriel, pour certains exercices, les modèles sont disponibles en téléchargement.



I.4 Analyse d'un modèle

Dans cette partie, nous allons analyser le modèle ultérieurement élaboré.

A partir d'un tel modèle, nous pouvons appliquer trois types d'analyse, toutes disponibles à partir du menu **Tools** :

1. Des outils de simulation permettant de calculer l'ordonnancement du jeu de tâches et de déterminer, à partir de cette simulation, divers critères de performance comme le respect des échéances, les temps de réponse (pire ou moyen), les temps de blocage sur ressources partagées par exemple.
2. Divers tests de faisabilité permettant de démontrer l'ordonnançabilité du jeu de tâches, en calculant les pires temps de réponse, le taux d'occupation des processeurs, ...
3. Des outils permettant de modifier le modèle d'architecture (dans un objectif d'exploration d'architectures) si ce dernier ne conduit pas au respect des échéances des tâches : algorithmes pour affecter les priorités des tâches, pour placer les tâches sur les processeurs ou modifier leurs paramètres selon leurs dépendances par exemple.

Sous les menus, quelques boutons offrent des raccourcis pour certaines analyses fréquemment employées. Ainsi, le bouton  déclenche le calcul et l'affichage d'une simulation alors que le bouton  provoque l'application de tests de faisabilités classiques sur le modèle. Il s'agit principalement du calcul des pires temps de réponse pour des modèles de tâches périodiques selon une méthode similaire à celle de Joseph et Pandia [JOS 86]. Pour des tests de faisabilité plus spécifiques (ex: transaction), ou une configuration des simulations, il est nécessaire d'utiliser les outils proposés par le menu **Tools**.

Les figures 6 et 7 présentent une copie d'écran, pour le modèle édité dans la partie I.2 après la pression des boutons  et . On y constate que les pires temps de réponse calculés par le test de faisabilité de Joseph et Pandia [JOS 86] ainsi que la simulation sur l'intervalle de faisabilité nous indiquent que toutes les tâches respectent leurs échéances. Le test de faisabilité sur le taux d'occupation du processeur n'est pas appliqué ici car l'outil ne peut déterminer comment les priorités ont été affectées ; pour ce faire, il aurait été nécessaire de choisir un ordonnanceur différent lors de l'édition de **core1** : l'ordonnanceur (e.g. **Rate Monotonic Protocol**).

Notez que dans les chronogrammes, les rectangles rouges verticaux représentent les instants de réveil des tâches alors que les rectangles horizontaux représentent les moments où les tâches s'exécutent. De nombreux paramètres d'affichage ou de simulation peuvent être modifiés via le menu **Tools/Scheduling/Scheduling Options**.

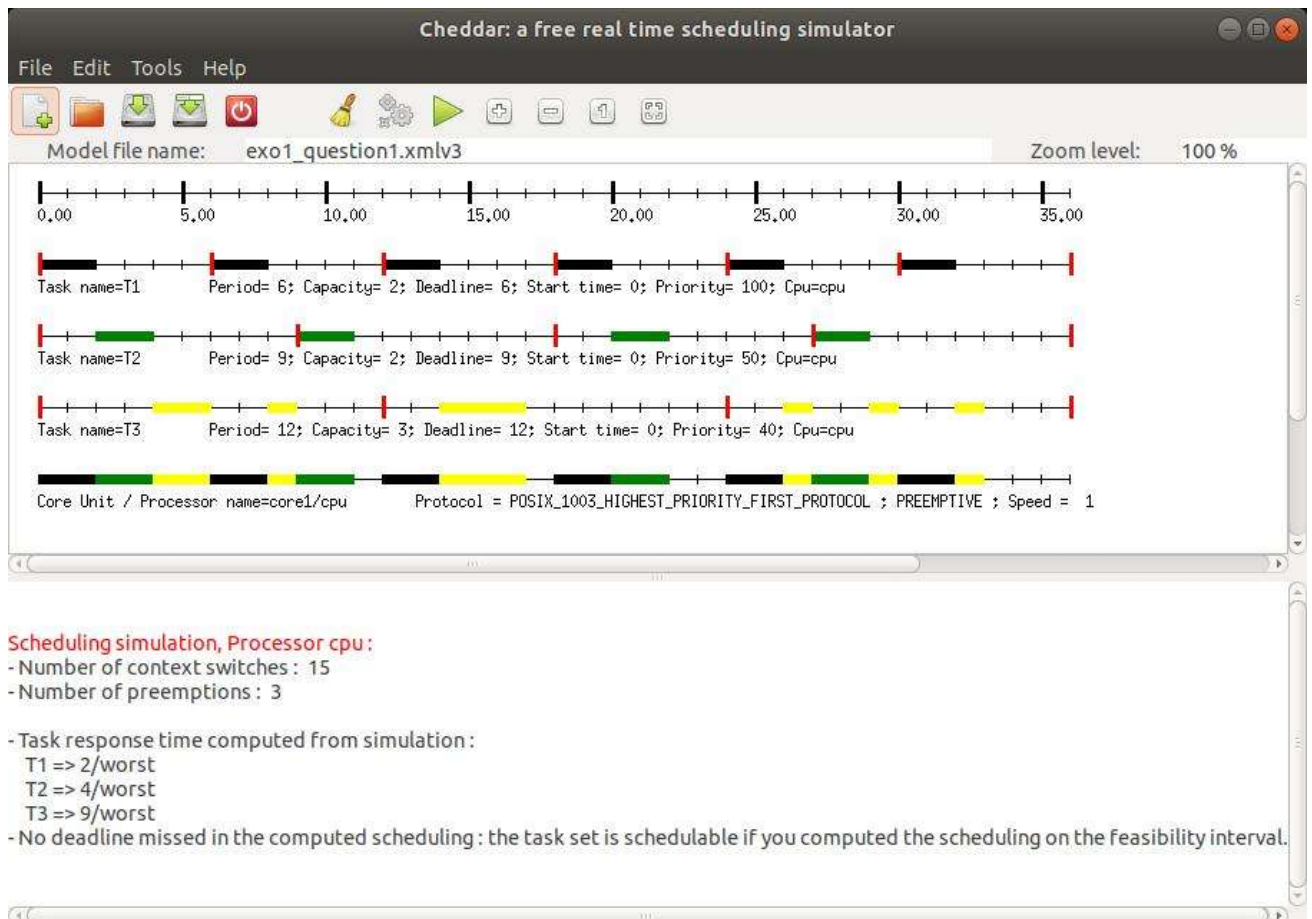


Fig. 6 : Simulation de l'ordonnancement

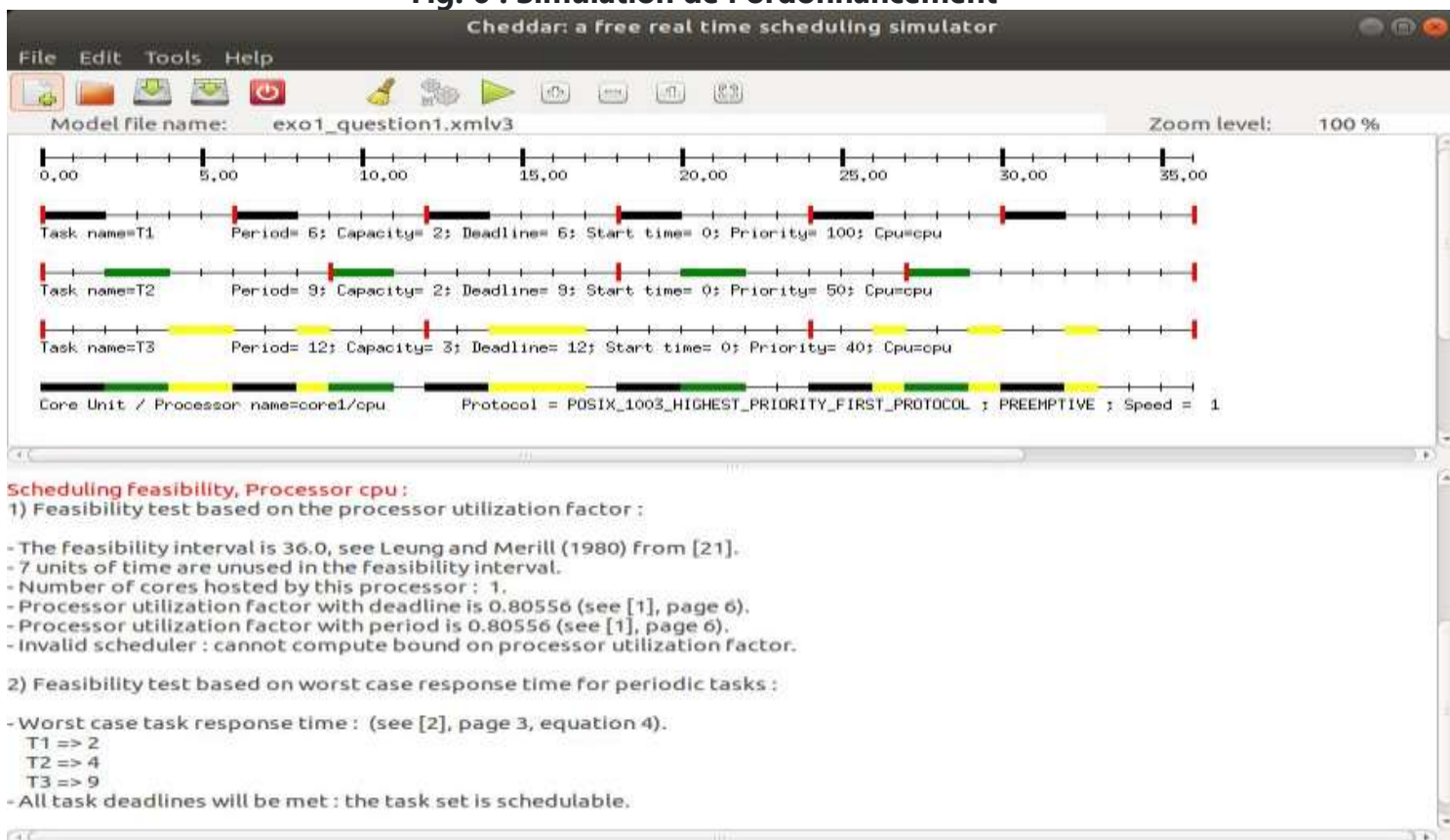





Fig. 7 : Analyse par tests de faisabilité

Exercice 1

1. Éditer un modèle Cheddar décrivant le système analysé dans les figures 6 et 7, puis, avec les boutons  et , vérifier que vous obtenez bien les résultats ci-dessus.
2. A partir de la simulation, lire les temps de réponse obtenus pour chaque tâche et pour chaque réveil d'une tâche (pour rappel, temps de réponse = date de terminaison - date de réveil correspondante). Pour une tâche donnée, le temps de réponse varie-t-il dans le temps ? Pourquoi ? Quand le temps de réponse d'une tâche a-t-il le plus de chance d'être le plus long ? Pourquoi ?
3. Modifier les paramètres des tâches de la façon suivante :
 - Tâche T1 : **Capacity** = 2, **Deadline** = **Period** = 6, **Start time** = 0
 - Tâche T2 : **Capacity** = 8, **Deadline** = **Period** = 24, **Start time** = 0
 - Tâche T3 : **Capacity** = 4, **Deadline** = **Period** = 12, **Start time** = 0
 - Priorités des tâches : appliquer l'algorithme Rate Monotonic.

Il s'agit d'un jeu de tâche dit harmonique (car chaque période du jeu de tâche est multiple des autres) et qui occupe 100% du processeur. Avec un jeu de tâche harmonique, un ordonnancement préemptif à priorité fixe et une affectation de priorité par Rate Monotonic, il est possible de trouver un ordonnancement respectant toutes les échéances au-delà des 69% de temps processeur occupé, jusqu'à atteindre un processeur occupé à 100%.

4. Avec Cheddar, générer un ordonnancement sur l'intervalle de faisabilité (bouton ). A partir de cette simulation, vérifier que ce jeu de tâches est effectivement ordonnançable.

II. Comparaisons d'algorithmes



Dans les exercices suivants, nous allons expérimenter et comparer 3 algorithmes d'ordonnancement temps réel classiques.

Exercice 2

Soient deux tâches T1 et T2, périodiques, synchrones et à échéances sur requêtes, définies par les paramètres suivants:

- Tâche T1 : **Capacity** = 4, **Deadline** = **Period** = 8, **Start time** = 0
- Tâche T2 : **Capacity** = 5, **Deadline** = **Period** = 10, **Start time** = 0
- Priorités des tâches : appliquer l'algorithme Rate Monotonic.

1. Charger le modèle du système pour l'exercice 2 : [ex2_base.xmlv3](#). Il se compose d'un processeur composé d'un seul coeur.

2. Modifier le modèle dans Cheddar pour ce jeu de tâches, s'exécutant sur un processeur (comportant un coeur) et utilisant un algorithme à priorité fixe préemptif.
3. Avec Cheddar, générer un ordonnancement sur l'intervalle de faisabilité (bouton )
4. A partir de la simulation, lire les pires temps de réponse obtenus pour chaque tâche. Existe-t-il des échéances manquées ?
5. Modifier votre modèle afin d'utiliser un algorithme EDF préemptif (valeur `Earliest_Deadline_First_Protocol` pour l'attribut `Scheduler type`). cette fois, puis refaire les questions 2 et 3.
6. Comparez les résultats obtenus. Que constatez-vous ? Est-ce surprenant ?
7. Modifiez votre modèle en conservant un ordonnanceur EDF préemptif mais avec l'ensemble de tâches ci-dessous. Notez que ce jeu de tâches comporte une tâche apériodique. Pour notre tâche apériodique, nous ne souhaitons pas nécessairement garantir le respect de son échéance. Pour définir une tâche apériodique avec Cheddar, il est nécessaire de modifier l'attribut `Task Type` (cf. figure 5).
 - Tâche périodique T1 : `Period=Deadline=8, Capacity=4, Start time = 0`
 - Tâche périodique T2 : `Period=Deadline=16, Capacity=8, Start time = 0`
 - Tâche **apériodique** T3 : `Deadline=15, Capacity=4, Start time = 0`
8. Ré-exécuter une simulation sur les 30 premières unités de temps (bouton )
Que constatez-vous ?
9. Qu'en concluez-vous sur l'adéquation d'EDF pour les systèmes temps réel critiques ? Est-ce le cas avec un ordonnancement à priorité fixe ?


Exercice 3

Soient deux tâches T1 et T2, périodiques, synchrones et à échéances sur requêtes, définies par les paramètres suivants:

- Tâche T1 : `Period=Deadline=9, Capacity=4, Start time = 0`
- Tâche T2 : `Period=Deadline=8, Capacity=3, Start time = 0`

On se propose maintenant d'étudier un nouvel algorithme d'ordonnancement: l'algorithme LLF (Least Laxity First). Ce dernier effectue l'élection des tâches prêtes grâce à leur laxité. La laxité d'une tâche est une information dynamique, i.e. qui évolue dans le temps. La laxité $Li(t)$ d'une tâche i à l'instant t s'évalue par $Li(t) = \text{Deadline} - \text{reste}(t)$ où $\text{reste}(t)$ est le reliquat de capacité à exécuter à l'instant t .

1. Charger le modèle du système pour l'exercice 3 : [ex3_base.xmlv3](#). Il se compose d'un processeur composé d'un seul coeur.
2. Construire un modèle avec Cheddar, pour ce jeu de tâches, s'exécutant sur un processeur comportant un coeur et utilisant un algorithme EDF préemptif.

3. Avec Cheddar, générer un ordonnancement sur l'intervalle de faisabilité (bouton ).
4. A partir de la simulation, lire les pires temps de réponse obtenus pour chaque tâche. Existe-t-il des échéances manquées ?
5. Modifier votre modèle afin d'utiliser un algorithme LLF préemptif cette fois ci, puis, refaire les questions 2 et 3.
6. Quelles différences peut-on constater sur les résultats obtenus par les deux algorithmes ?
7. Conclure sur le choix entre ces deux algorithmes.

III. Partage de ressources

Dans les exercices précédents, nous supposons les tâches indépendantes. Par la suite, nous allons illustrer les dépendances de tâches avec une dépendance classique : le partage de ressources.

Exercice 4

Soient trois tâches périodiques, synchrones et à échéances sur requêtes, définies par les paramètres suivants:

- Tâche T1 : **Period=Deadline=6, Capacity=2, Start time = 0**
- Tâche T2 : **Period=Deadline=8, Capacity=2, Start time = 0**
- Tâche T3 : **Period=Deadline=12, Capacity=5, Start time = 0**

On utilise un algorithme à priorité fixe préemptif pour ordonnancer les tâches. Les priorités sont affectées selon Rate Monotonic.

Les tâches T1 et T3 se partagent une ressource dont le nom est S. T1 et T3 accèdent à S en exclusion mutuelle.

- T3 accède à la ressource durant la totalité de sa capacité.
 - T1 accède à la ressource durant la deuxième unité de temps de sa capacité uniquement.
1. Charger le modèle du système pour l'exercice 4 : [ex4_base.xmlv3](#). Il se compose d'un processeur composé d'un seul coeur et le jeu de tâches décrit ci-dessus.
 2. Vous devrez ajouter la ressource partagée grâce au menu **Edit/Software/Resource** qui affiche la fenêtre suivante :

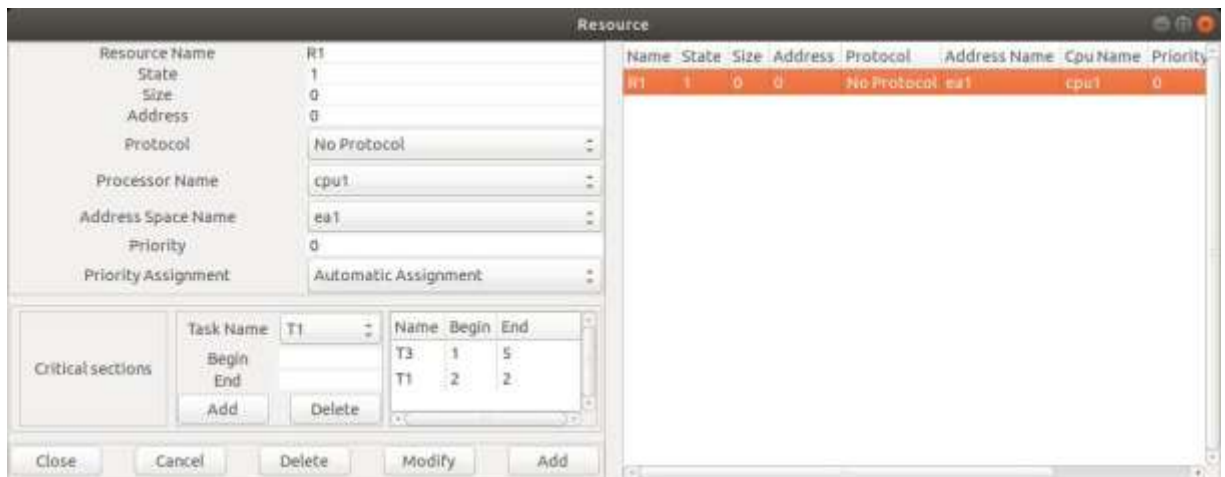



Fig. 8 : Édition d'une ressource partagée

Dans cette fenêtre et pour nos exercices, une ressource sera définie par les attributs suivants:

- **Name** : le nom unique de la ressource.
- **CPU Name** et **Address space name** : comme pour les tâches, ces attributs indiquent le placement de la ressource.
- **State** : l'état de la ressource. Une ressource fonctionne ici comme un sémaphore à compteur. Une ressource possède un compteur entier. Si ce compteur est inférieur ou égal à zéro, une tâche qui demande l'allocation de la ressource sera bloquée. Si une tâche libère la ressource, le compteur est incrémenté. Si une tâche alloue la ressource, le compteur est décrémenté. Pour nos exercices, **State** doit être positionné à 1.
- **Protocol** : le protocole à appliquer lors des accès à la ressource. La valeur **No Protocol** indique que le simulateur n'effectuera aucun changement de priorité lors des accès à la ressource : un sémaphore FIFO est alors utilisé. Dans cet exercice et le suivant, nous utiliserons aussi les valeurs **Priority Inheritance Protocol** (ou **PIP**) et **Priority Ceiling Protocol** (ou **PCP**). Pour l'instant, positionner la valeur **No Protocol** pour la ressource S.
- **Priority assignment** : qui indique s'il faut manuellement calculer la priorité plafond d'une ressource ou non. Pour nos exercices, il est nécessaire de laisser cet attribut à la valeur **Automatic Assignment**.
- **Sections critiques** : dans cette partie sont spécifiés les instants de prise et de libération de la ressource pour chaque tâche. **Begin** indique le début d'une section critique et **End** la fin d'une section critique. Les boutons **Add** et **Delete** permettent d'ajouter ou de supprimer une section critique associée à la ressource. Ainsi, dans la figure 8, il est spécifié que T1 alloue/requiert la ressource au début de la seconde unité de temps de sa capacité et la libère à la fin de cette même unité

de temps alors que T3 requiert la ressource avant d'exécuter la 1ère unité de temps de sa capacité et libère la ressource après la cinquième unité de temps.

3. Calculer l'ordonnancement avec Cheddar sur l'intervalle de faisabilité (bouton ). Vous devez obtenir l'ordonnancement ci-dessous (figure 9) :

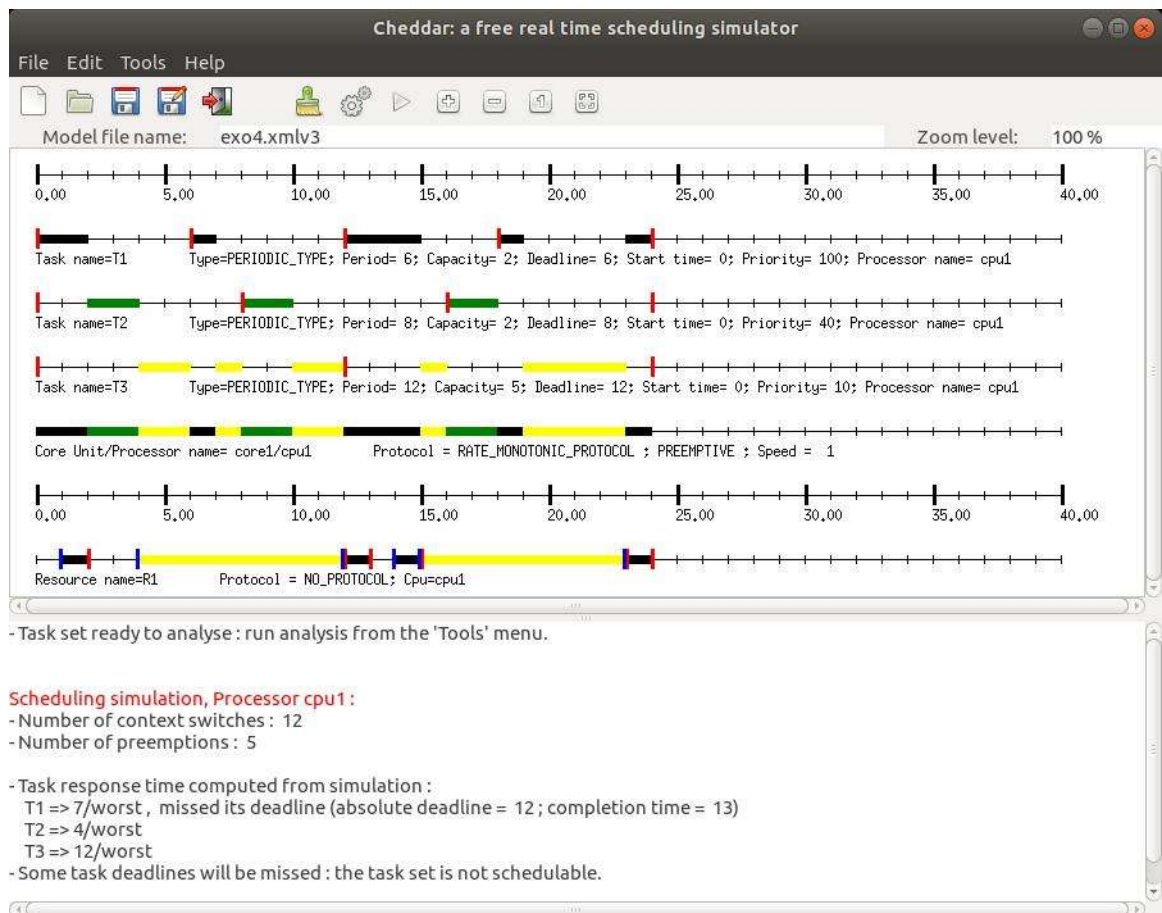



Fig. 9 : Ordonnancement avec une ressource partagée

4. Sur le chronogramme de chaque ressource, les rectangles horizontaux indiquent quand la ressource est utilisée : la couleur de ces rectangles correspond à la couleur de la tâche qui utilise la ressource. Les rectangles bleus verticaux indiquent les instants où une ressource est allouée par une tâche. Les rectangles rouges verticaux indiquent les instants où une ressource est libérée par une tâche.
5. Indiquer à quels instants les tâches T1 et T3 allouent et libèrent la ressource S dans ce chronogramme.
6. Cet ordonnancement comporte-t-il des inversions de priorité ? Où ?
7. Changer le protocole de S par **Priority Inheritance Protocol** (ou **PIP**), puis, calculer à nouveau l'ordonnancement. Avec PIP, une tâche qui bloque une autre plus prioritaire qu'elle, exécute la section critique avec la priorité de la tâche bloquée.

8. Que constatez-vous concernant l'inversion de priorité ?
9. Indiquer à quels instants la priorité des tâches change.
10. Avec le bouton , calculer les pires temps de réponse avec la méthode de Joseph et Pandia. Cette méthode suppose que les tâches soient indépendantes. Comparer avec les temps de réponse obtenus lors de la simulation. Que constatez vous ? Expliquez ces différences.
11. A partir de la simulation, donnez le temps de blocage subi par chaque tâche à cause de la ressource S.

Exercice 5

Nous regardons un exemple de modèle Cheddar utilisant deux ressources cette fois-ci. Soient deux tâches périodiques et à échéances sur requêtes, définies par les paramètres suivants:


- Tâche T1 : **Period=Deadline=31, Capacity=8, Start time = 0**
- Tâche T2 : **Period=Deadline=30, Capacity=8, Start time = 2**
- Attention, vous noterez que ces tâches ne sont pas synchrones (c.f. valeurs des attributs **Start time**) .
- L'affectation des priorités est réalisée selon Rate Monotonic.

On utilise un algorithme à priorité fixe préemptif pour ordonnancer ces tâches.

Les tâches T1 et T2 accèdent à 2 ressources partagées, les ressources R1 et R2 :

- T1 accèdent à R1 de la 2^{ème} unité de temps de sa capacité à la 8^{ème} incluse.
- T2 accèdent à R1 de la 6^{ème} unité de temps de sa capacité à la 8^{ème} incluse.
- T1 accèdent à R2 de la 4^{ème} unité de temps de sa capacité à la 8^{ème} incluse.
- T2 accèdent à R2 de la 2^{ème} unité de temps de sa capacité à la 8^{ème} incluse.

On suppose que les ressources R1 et R2 utilisent le protocole **PIP (Priority Inheritance Protocol)**.

1. Charger le modèle du système pour l'exercice 5 : [ex4_base.xmlv3](#). Il se compose d'un processeur composé d'un seul coeur et le jeu de tâches décrit ci-dessus.
2. Ajouter R1 et R2, puis, calculer l'ordonnancement sur les 30 premières unités de temps par simulation (bouton ).
3. Indiquer à quels instants les tâches T1 et T2 verrouillent et déverrouillent les ressources R1 et R2 dans ce chronogramme.
4. Que se passe-t-il de remarquable ? Est-ce surprenant ?
5. On suppose maintenant que l'on applique le protocole PCP (**Priority Ceiling Protocol**) dans la fenêtre de la figure 8). Il faudra modifier la définition des ressources R1 et R2 pour ce faire. PCP fonctionne de la façon suivante :

- Un plafond de priorité est associé à chaque ressource. Cette priorité est égale à la plus grande priorité des tâches qui accèdent à la ressource. Cette information est statique et va être automatiquement calculée par Cheddar.
 - Une tâche qui bloque une autre plus prioritaire qu'elle, exécute la section critique avec la priorité de la tâche bloquée (héritage de priorité comme dans PIP).
 - Lorsqu'une tâche tente d'allouer une ressource, elle reste bloquée si sa priorité actuelle n'est pas strictement supérieure à tous les plafonds de priorité des ressources précédemment allouées par d'autres tâches. C'est cette nouvelle condition de blocage qui constitue la différence essentielle entre PIP et PCP.
6. Recalculer le chronogramme sur les 30 premières unités de temps lorsque les ressources sont gérées grâce à PCP.
 7. Indiquer à quels instants les tâches T1 et T2 verrouillent et déverrouillent les ressources R1 et R2 dans ce chronogramme.
 8. Indiquer à quels instants la priorité des tâches change.
 9. Comparer cette simulation avec celle obtenue lors de la question 1.

IV. Exercices de synthèse

Exercice 6

On étudie dans cet exercice un système de visualisation intégré dans un véhicule automobile. Ce système est constitué d'un ensemble de tâches dont on souhaite étudier le respect de leurs contraintes temporelles.

Cette application est constituée de 5 tâches périodiques :

1. La tâche CAPTEUR_1 qui lit toutes les 10 millisecondes la vitesse du véhicule.
2. La tâche CAPTEUR_2 qui lit toutes les 10 millisecondes la température dans l'habitacle du véhicule.
3. La tâche CAPTEUR_3 qui lit toutes les 40 millisecondes la position GPS du véhicule.
4. La tâche AFFICHAGE_1 qui produit toutes les 12 millisecondes, un résumé des informations produites par les tâches CAPTEUR_1, CAPTEUR_2 et CAPTEUR_3 sur un écran LCD.
5. La tâche AFFICHAGE_2 qui affiche à la demande de l'utilisateur une carte routière. L'affichage doit être réactualisé toutes les 6 millisecondes.

Après une campagne de mesure des programmes implantés par les tâches sur la cible, on détermine que :

1. Les tâches CAPTEUR_2 et AFFICHAGE_1 ont une durée d'exécution ne dépassant pas 2 millisecondes.
2. La tâche CAPTEUR_3 requiert 2 à 4 millisecondes pour exécuter une de ses activations.
3. Enfin, les tâches CAPTEUR_1 et AFFICHAGE_2 ont une durée d'exécution inférieure à une milliseconde.

On suppose que toutes les tâches démarrent à un instant identique. Enfin, on souhaite que les tâches terminent une activation donnée avant le début de leur activation suivante.

Le système utilisé propose un ordonnanceur préemptif à priorité fixe comprenant 32 niveaux de priorité (de 0 à 31). 0 est le niveau de priorité le plus faible. Le système ne permet pas l'affectation d'une priorité identique à plusieurs tâches. Le système comporte un processeur avec un coeur uniquement.

Question 1 :

- En général, quels sont les critères utilisés pour fixer la priorité des tâches ?
- Déterminer les différents paramètres des tâches nécessaires à leur analyse d'ordonnancement. Vous motiverez plus particulièrement le choix de la priorité.

Question 2 :

A partir des priorités déterminées dans la question précédente, calculer avec Cheddar les pires temps de réponse des tâches CAPTEUR_1, CAPTEUR_2, et CAPTEUR_3 avec la méthode de Joseph et Pandia.

Question 3 :

En fait, les tâches CAPTEUR_1, CAPTEUR_2, et CAPTEUR_3 partagent une zone de mémoire accédée en exclusion mutuelle. CAPTEUR_1 et CAPTEUR_2 accèdent à cette ressource pendant toute leur capacité. CAPTEUR_3 accède à la ressource pendant les deux premières millisecondes de sa capacité. Le sémaphore qui protège la zone de mémoire partagée applique le protocole PIP. Expliquer pourquoi les temps de réponse calculés dans la question précédente ne constituent plus des pires cas.

Question 4 :

Avec Cheddar, calculer le chronogramme décrivant l'ordonnancement des tâches pendant les 60 premières milli-secondes. Vous signalerez les instants où la ressource est allouée et libérée, les éventuelles inversions de priorité ainsi que les instants où une tâche hérite d'une priorité.

Question 5 :

Pour les questions 5 et 6, nous supposons maintenant que les tâches ne partagent plus de ressource : les tâches sont donc indépendantes.

Afin d'activer les tâches plus fréquemment, on souhaite tester une configuration où les tâches sont placées sur deux processeurs (les processeurs **a** et **b**). Chaque processeur comporte un coeur uniquement : il n'y a donc pas de migration des tâches d'un coeur à un autre.

Les tâches sont maintenant définies par les paramètres suivants :

- Tâche AFFICHAGE_1 : Processeur = **b** ; **Capacity** = 4, **Period**=**Deadline**= 6
- Tâche CAPTEUR_1 : Processeur = **b** ; **Capacity** = 2, **Period**=**Deadline**= 20
- Tâche AFFICHAGE_2 : Processeur = **a** ; **Capacity** = 2, **Period**=**Deadline**= 3
- Tâche CAPTEUR_2 : Processeur = **a** ; **Capacity** = 2, **Period**=**Deadline**= 5
- Tâche CAPTEUR_3 : Processeur = **a** ; **Capacity** = 2, **Period**=**Deadline**= 5

Les tâches sont toujours ordonnancées selon un algorithme à priorité fixe. De plus, les priorités sont affectées selon l'algorithme Rate Monotonic.

Montrer, sans simuler l'ordonnancement de ce système, qu'il existe au moins une des tâches qui ne respecte pas ses contraintes temporelles.

Question 6 :

En fait, l'affectation des tâches aux processeurs **a** et **b** a été réalisée au hasard. En effet, toutes les tâches peuvent à la fois s'exécuter sur le processeur **a** ou sur le processeur **b**. Les processeurs diffèrent uniquement par leur vitesse d'exécution : le processeur **b** est 2 fois plus rapide que le processeur **a**.

Affecter les tâches aux processeurs **a** et **b** de sorte que les contraintes temporelles de toutes les tâches soient respectées. **Justifiez vos choix.**

Exercice 7

Cet exercice est extrait de [COT 00] et consiste à diagnostiquer et corriger un modèle d'architecture illustrant celui du système d'exploration de la mission Mars Pathfinder. En 1997, la mission Mars PathFinder a déployé le robot mobile Sojourner sur Mars. La mission de ce robot et de sa sonde était contrôlée par un logiciel multitâches s'exécutant sur un système VxWorks. Ce logiciel peut être modélisé par l'ensemble de tâches suivant :

Nom des tâches	Priorités	Périodes/Echéances	Temps d'exécution
ORDO_BUS	1	125 ms	25 ms
DONNEES	2	125 ms	25 ms
PILOTAGE	3	250 ms	25 ms
RADIO	4	250 ms	25 ms
CAMERA	5	250 ms	25 ms
MESURE	6	5000 ms	50 ms
METEO	7	5000 ms	50 ms ou 75 ms

- Toutes les tâches sont périodiques, synchrones et à échéances sur requêtes.
 - La tâche METEO est activée, parfois pour un traitement de 50 ms, parfois pour un traitement de 75 ms selon la quantité de données à transmettre.
 - Les priorités indiquées ci-dessus sont des priorités VxWorks. Avec VxWorks, les niveaux de priorités vont de 0 à 255 sachant que le niveau de priorité le plus fort est 0 (attention, c'est l'inverse avec Cheddar car 0 est le niveau de priorité le plus faible et 255 le plus fort).
 - Les tâches DONNEES, PILOTAGE, MESURE et METEO utilisent en exclusion mutuelle une ressource partagée durant tout leur temps d'exécution. Pour des raisons d'économie, l'accès en exclusion mutuelle à cette ressource est implanté grâce à un sémaphore qui n'implante pas de protocole d'héritage de priorités comme PIP ou PCP.
1. Pendant le déroulement de la mission Mars PathFinder, les opérateurs de la NASA ont constaté que certaines échéances de ce logiciel ont été manquées, ce qui a provoqué plusieurs redémarrages du robot ainsi que la perte de données scientifiques. A partir de votre modèle, indiquez quelles sont les échéances qui ne sont pas respectées ? Pourquoi ?
 2. Proposez une solution à ce problème.

V. Références

- [BUR 97] A. Burns and A. Wellings. Real-time Systems and Programming Languages. Addison Wesley, 1997.
- [COT 00] F. Cottet, J. Delacroix, C.Kaiser, and Z. Mammeri. Ordonnancement temps réel. Hermès, 2000.
- [DEM 99] I. Demeure and C. Bonnet. Introduction aux systèmes temps réel. Collection pédagogique de télécommunications, Hermès, Septembre 1999.
- [JOS 86] M. Joseph and P. Pandya. Finding Response Time in a Real-Time System. Computer Journal, 29(5):390--395, 1986.
- [KRI 97] C.M. Krishna and K.G. Shin. Real-Time Systems. Mc Graw-Hill International Editions, 1997.

Last Updated by Frank Singhoff (singhoff@univ-brest.fr)

Last update: December 2021

Frank Singhoff (singhoff@univ-brest.fr), **Hai Nam Tran** (hai-nam.tran@univ-brest.fr)

École d'Été Temps Réel (ETR), Poitiers, Futuroscope, 20-24 September, 2021

TP également assuré lors d'ETR 2015 et 2017.