Plate-forme d'expérimentation pour l'ordonnancement des applications temps réel à contraintes strictes
L. David, E. Grolleau and S. Constantin., Acte de la Conf. Real Time Systems (RTS'02), Ed. Teknea, Paris, France, 2002, pp. 33-46

Résumé : Nous étudions à partir d'une plate-forme de test l'apport des techniques d'ordonnancement hors-ligne. Le procédé étudié pour la plate-forme est à fortes contraintes de temps, puisqu'il s'agit de maintenir un pendule en position verticale inversée. Pour cette étude, nous utilisons le noyau temps réel VxWorks$^{(TM)}$. L'étude est menée de la spécification/conception du système de commande du chariot, jusqu'à la phase d'ordonnancement de l'application à l'aide d'une méthode d'ordonnancement hors-ligne et sa mise en oeuvre. Nous proposons une solution permettant de diminuer la durée du quantum de temps au-delà des possibilités usuelles des systèmes d'exploitation temps réel afin d'ordonnancer finement le système de tâches et ainsi diminuer les temps de réponse du système.
Mots clés : Système temps réel, ordonnancement en ligne et hors ligne, VxWorks, SA-RT, DARTS, granularité des ordonnanceurs.

1 Introduction

Les techniques d'ordonnancement temps réel se classent en deux catégories. On distingue d'une part l'approche en-ligne, basée sur une attribution fixe [LL73, LW82, Aud91] ou variable des priorités des tâches du système [Der74, Lab74, DM89]. A chaque instant, c'est la tâche de plus forte priorité qui se voit attribuer le processeur. D'autre part, on distingue l'approche hors-ligne qui se base sur une construction a priori d'une séquence d'ordonnancement valide [XP92, Gro99]. Chaque approche présente des avantages et des inconvénients. Il existe ainsi des algorithmes d'ordonnancement en-ligne optimaux dans certains contextes lorsque les tâches sont indépendantes [Der74, Lab74, DM89]. Des tests analytiques simples permettent aussi de conclure à l'ordonnançabilité de tels systèmes lorsque les tâches sont simultanées [LL73, BHR90, Aud91]. Cependant, dès lors que certaines tâches sont différées, l'étude d'ordonnançabilité est co-NP-difficile [LW82, BHR90]. En outre, si certaines tâches partagent des ressources, il n'existe alors aucune solution polynomiale optimale [Mok83] au problème de l'ordonnancement, et malgré l'introduction de proto-

coles de gestion de ressources [SRL90, CL90, Bak91], il n'existe que des conditions suffisantes d'ordonnançabilité. C'est précisément pour ce type de problème que différentes approches hors-ligne ont été proposées [XP92, Gro99].

Dans le souci de pouvoir mettre en pratique ces techniques d'ordonnancement pour les applications temps réel, nous proposons dans cette étude la description d'une plate-forme de test des mécanismes d'ordonnancement basés sur le modèle de tâches, communément désigné par celui de Liu et Layland [LL73]. Dans ce projet, nous sommes confrontés au manque de déterministe temporel des ordinateurs actuels [Tör98], mettant le plus souvent en relief leur capacité de calculs et leur rapidité à les mener. Pour une application temps réel pilotant un procédé, ceci peut être contradictoire avec les impératifs de validation temporelle : il ne s'agit pas en effet de contrôler le procédé rapidement, mais de s'adapter à la dynamique de son évolution [CNR88]. Pour la plupart des applications temps réel, la vitesse des ordinateurs actuels est telle que la durée d'exécution des tâches formant l'application peut se rapporter à la durée d'une unité de temps de l'ordonnanceur du noyau temps réel, qui doit se situer selon la norme POSIX [Gal95] dans l'intervalle $10 \sim 20ms$. Dans certains cas également, la durée de l'ensemble des tâches peut même se placer dans une seule unité de temps. Devant ce problème récurrent qui écarte ou minimise l'étude d'ordonnancement et qui sous exploite la ressource processeur, nous proposons une technique permettant d'affiner cette granularité de l'ordonnanceur, et de décrire une configuration de tâches compatible avec les techniques d'ordonnancement classiques.

Dans une première partie, nous présentons la plate-forme de test des techniques d'ordonnancement puis nous décrivons les phases de spécifications et de conception d'une application contrôlant le procédé test. Pour cela, nous utilisons la méthode de spécification SA-RT [WM86], suivie d'une conception DARTS [Gom84]. La seconde partie décrit les techniques utilisées pour extraire un modèle de tâches à ordonnancer à partir de l'implémentation, puis présente en détail une solution d'ordonnancement basée sur l'approche hors ligne.

2 Plate-forme de test : le pendule inversé

La plate-forme de test utilisée pour mettre en pratique les techniques d'ordonnancement, est basée sur l'expérience du pendule inversée. Celle-ci consiste à piloter un chariot pouvant se mouvoir sur un rail, de sorte qu'une tige métallique - le pendule -, montée en rotation libre sur le chariot, puisse être maintenue en position verticale inversée (cf. figure 1). Cette position étant instable, une réaction trop tardive dans la correction du mouvement du chariot provoque la chute du pendule, avec l'impossibilité de corriger sa trajectoire (moteur à puissance finie, inertie, etc...). En outre, les principes généraux du traitement du signal, ainsi que la nécessité d'obtenir un système stable au sens automatique du terme, imposent une forte régularité d'exécution pour les processus d'interfaçage entre le procédé et le système numérique. Ces différentes contraintes de temps font de ce procédé un support d'étude

intéressant.

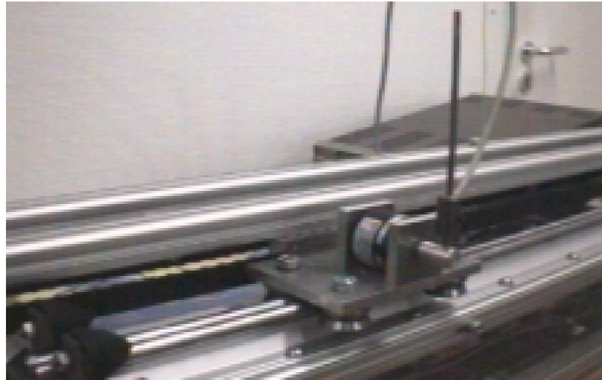

FIG. 1 – Le pendule en“position inversée” sur son chariot.

2.1 Description de la plate-forme

Le mouvement du chariot est obtenu par un moteur à courant continu qui constitue le seul actionneur de cette plate-forme. Les capteurs sont, quant à eux, au nombre de 4. Un codeur incrémental situé sur l'axe de rotation libre du pendule détermine l'angle $\theta(t)$ que fait ce dernier avec la verticale. Deux capteurs de fin de course situés de part et d'autre du rail indiquent si le chariot est en butée. Enfin, un potentiomètre rotatif permet d'indiquer la position du chariot sur le rail.

Plusieurs phases de fonctionnement du pendule peuvent être distinguées : une phase d'initialisation, pendant laquelle le chariot est amené au centre du rail et pendant laquelle on attend que le pendule se soit immobilisé en position verticale descendante ; une phase d'oscillations du chariot, au cours de laquelle, par des mouvements de va-et-vient du chariot, on amène le pendule dans une position verticale ascendante ; et enfin, une phase de maintien en position inversée. Dans un souci de simplification, seule cette dernière phase est étudiée dans cet article. En outre, pour maintenir cette position, nous utilisons la notion de boucle d'asservissement, en reliant alors le mouvement à impulser au chariot, à l'angle que fait le pendule avec la verticale.

Le système d'exploitation temps réel utilisé est VxWorks. Ce système est conforme à la norme POSIX, mais toutes les fonctions implémentées selon POSIX sont doublées par des fonctions propriétaires annoncées par WindRiver comme étant plus efficaces et plus souples d'utilisation. Dans ce projet, nous avons finalement choisi d'utiliser les primitives WindRiver afin d'accéder à certaines fonctionnalités facilitant la validation : pour les sémaphores par exemple, WindRiver ajoute par rapport à la norme POSIX le protocole d'héritage de priorités à priorité plafond. Associé à ce système d'exploitation, WindRiver fournit aussi un Environnement de Développement Intégré (EDI), appelé Tornado, qui bénéficie de nombreuses fonctions de debogage, spécifiques pour certaines, aux applications temps réel. Afin de pouvoir

les utiliser pleinement, nous avons fait le choix d'une architecture de développement de type client-serveur[1]. Après une phase d'initialisation, l'ordinateur cible (ou client) télécharge ainsi un noyau temps réel complet contenant également le code de l'application (cf. figure 2)

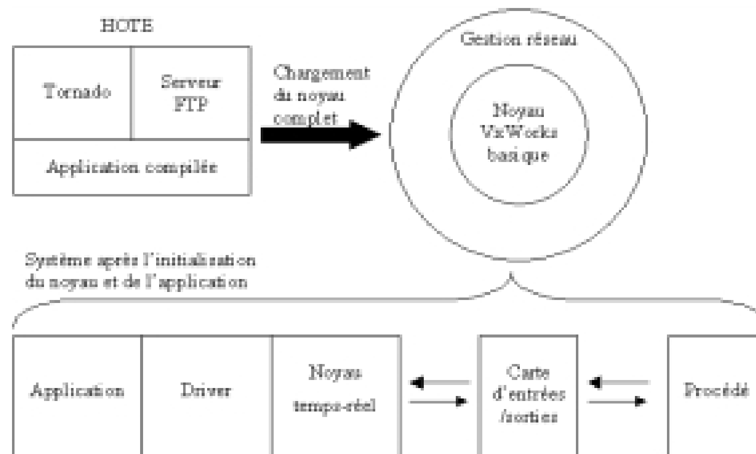

FIG. 2 – Initialisation et architecture du système temps-réel.

Une fois lancée, l'application dialogue avec le procédé par des appels de routines agissant sur la carte d'entrée/sortie et qui constitue l'interface entre l'ordinateur et la partie électronique de la plate-forme.

2.2 L'application de contrôle du procédé

L'application est développée à partir d'une spécification SA-RT, puis d'une conception DARTS. Le diagramme de contexte, qui illustre les interactions entre le procédé et le système numérique (cf figure 3), est défini à partir de l'ensemble des capteurs et actionneurs du système temps réel étudié. Il introduit également la représentation des flots de données entrants et sortants de l'application. Le processus fonctionnel "Commander moteur" reçoit ainsi des flots de données continus de la part du capteur d'angles et du potentiomètre de position. Il reçoit aussi un flot de données discret provenant des capteurs de butées. Puis en sortie, le processus fonctionnel envoie des flots de données continus vers un écran, et principalement vers le moteur. Deux flots de contrôle portant les événements "marche" et "arrêt" ont été ajoutés en provenance d'une console.

Une fois l'application spécifiée via la méthode SART, la méthode de conception DARTS (cf. figure 4) permet d'initier un découpage en tâches logicielles. Nous distinguons ainsi dans le diagramme, 7 tâches temps réel : 5 sont directement périodiques (flèche “HTR”) et 2 sont synchronisées par l'intermédiaire de boîte aux lettres (tâches “Moteur” et “Alarme” avec les tâches “PID” et “Contrôle butées”).

[1]On pourra néanmoins à terme, rendre l'application temps réel totalement indépendante de Tornado, et se diriger vers une application de type embarqué.

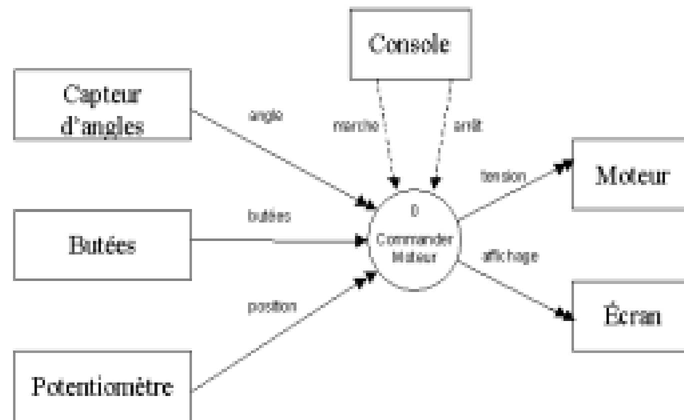

FIG. 3 – Diagramme de contexte de l'application.

Dans ce diagramme, nous distinguons également deux zones mémoires partagées en lecture/écriture, qui seront protégées par des sémaphores binaires.

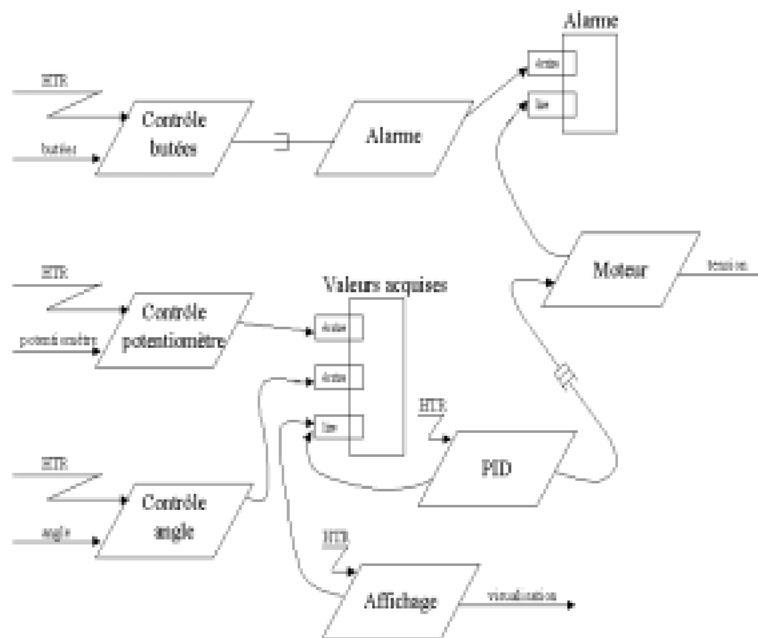

FIG. 4 – Diagramme DARTS.

Les tâches temps réel ainsi déterminées, l'application est implémentée en C dans l'environnement de développement Tornado. Après la phase de validation fonctionnelle, nous devons prendre en compte l'aspect comportemental de l'application, au travers de contraintes, comme les précédences, comme la régularité d'exécution de certaines tâches, ou encore comme le respect d'échéances. Cette prise en compte se traduit par des contraintes portant sur les paramètres temporels des tâches qui sont définis dans la partie suivante.

3 L'ordonnancement des tâches

3.1 Détermination des paramètres temporels

Dans le modèle de tâches de Liu et Layland, les paramètres temporels à définir se rapportent à une échelle des temps discrétisée, dont la plus petite durée correspond à la granularité de l'ordonnanceur. Bien que les contraintes temporelles déduites du procédé s'inscrivent dans une échelle de temps continue, elles doivent être exprimées dans une échelle discrétisée afin de pouvoir conduire l'étude d'ordonnancement. Sous VxWorks, la granularité de l'ordonnanceur est définie par la durée séparant deux "ticks" successifs, générés régulièrement par le noyau temps réel. La norme POSIX [Gal95] recommande une granularité comprise dans l'intervalle $10 \sim 20ms$; la société WindRiver va même au delà de cette norme et garantit le fonctionnement de VxWorks avec un quantum de temps de $1ms$. Au travers de notre plate-forme de test, nous verrons cependant que la fréquence de ces "ticks" reste bien faible au regard de la fréquence d'un processeur. Nous donnerons alors une solution permettant d'affiner la granularité de l'ordonnanceur et permettant de mener une étude d'ordonnançabilité complète.

3.1.1 La pire durée d'exécution d'une tâche

Méthode de Calcul : Pour effectuer les mesures temporelles, VxWorks possède un ensemble de fonctions capables de lire un compteur associé au processeur, et qui est incrémenté à chaque top d'horloge. La méthode de calcul utilisée reprend celle décrite dans [Sys99]. Elle consiste à encadrer chaque instruction à mesurer par les fonctions de lecture du compteur, conduisant ainsi à deux valeurs entières, qu'il suffit alors de soustraire pour obtenir la durée d'exécution de la tâche en cycle processeur. Afin d'obtenir une valeur fiable, la mesure est effectuée plusieurs fois (plusieurs centaines de millions de mesures) en utilisant une boucle. Il s'agit ensuite de convertir ces durées exprimées en cycle CPU, dans l'unité de temps choisie.

Problèmes rencontrés : L'ordinateur cible utilisé est un ordinateur grand public de type Pentium III, les mécanismes d'accélération des traitements des processeurs modernes (pipeline, caches), doivent donc être contournés. En effet, ces mécanismes peuvent induire une différence de durée d'exécution entre un traitement effectué à un instant t_1 et un autre effectué à un instant t_2 ($t_1 \neq t_2$). Nous désactivons ainsi les mémoires caches du processeur, puis nous plaçons une temporisation de plusieurs millisecondes juste avant la portion de code, dont on souhaite mesurer le temps d'exécution. En effet, en voulant effectuer plusieurs mesures de la durée d'exécution en un cycle, nous nous plaçons précisément dans une situation où les "pipelines" sont les plus efficaces. Cette temporisation permet alors de vider les "pipe-lines", puisque la durée du cycle processeur est de l'ordre de quelques nanosecondes, à comparer avec la durée de quelques millisecondes de la temporisation.

Un autre problème, d'ordre logiciel cette fois-ci, concerne les routines systèmes de très grandes priorité, qui peuvent venir perturber les mesures effectuées pen-

dant une boucle. Ces routines sont appelées via des interruptions systèmes, qu'il faut masquer le temps de la mesure. C'est le cas notamment de l'interruption qui génère le "tick" ordonnanceur. Cette méthode de masquage des interruptions reste cependant inefficace dans le cas d'une portion de code contenant des sémaphores. En effet, le fonctionnement des sémaphores est basé précisément sur le principe de masquage/démasquage des interruptions : lors de la libération d'un sémaphore, le démasquage des interruptions démasque non seulement les interruptions masqués par la prise du sémaphore, mais aussi celles déjà masquées auparavant. La solution que nous proposons dans ce cas est l'utilisation de l'instruction "taskdelay(1)" de VxWorks qui permet de suspendre une tâche en cours de traitement jusqu'au prochain "tick". Ainsi, dans le cas de la mesure de la durée d'exécution d'une prise ou d'une libération de sémaphore, nous utilisons cette technique pour contourner les interruptions régulières liées aux "ticks". Il est à noter que ce genre de mesure conduit à un ordre de grandeur de quelques nanosecondes, à comparer avec la durée entre deux "ticks", de l'ordre de la ms.

D'autres interruptions systèmes peuvent avoir lieu, mais nous savons les détecter puisqu'elles génèrent des durées d'exécution dix à vingt fois supérieures à la durée moyenne mesurée. Cette détection a été facilitée par l'outil de construction de Diagramme de Gantt, fourni avec l'environnement de développement Tornado/VxWorks. Finalement, nous retenons pour chacune des tâches, les durées, exprimées en cycle CPU, reportées dans le tableau 1.

tâche	butée	alarme	potentiomètre	angle	moteur	calcul_PID	affichage
c_i	2046	1030	9584	2853	4368	7092	29613

TAB. 1 – Durées des tâches de l'application (mesurées en cycle CPU).

Résultats provisoires : Nous devons maintenant rapporter ces durées à la granularité de l'ordonnanceur, en prenant en compte également les interruptions systèmes pouvant avoir lieu entre deux "ticks" successifs. Introduisons pour cela quelques notations :

- δ_{ut} la durée, en cycle CPU, séparant deux "ticks" successifs,
- δ_{MaxSys}, la durée maximum, en cycle CPU, des interruptions systèmes intervenant entre deux "ticks" successifs,
- et δ_d, la durée, en cycle CPU, disponible pour le traitement des tâches de notre application.

Nous avons alors la relation $\delta_d = \delta_{ut} - \delta_{MaxSys}$. Soit c_i, la durée en cycle CPU de la tâche τ_i provenant du tableau 1, on peut déterminer le nombre de cycles maximal nécessaire à l'exécution de la tâche c_{i_T}, en prenant en compte les interruptions systèmes pouvants'intercaler. La formule utilisée est récursive, c_{i_T} est le point fixe de la formule :

$$R_o = c_i$$
$$R_i = \left\lceil \frac{R_{i-1}}{\delta_{ut}} \right\rceil \cdot \delta_{MaxSys} + R_{i-1}$$

Cette durée est exprimée en cycle CPU. Enfin, nous introduisons la pire durée d'exécution C_i exprimée en nombre d'unité de temps, par la relation : $C_i = \left\lceil \frac{c_{i_T}}{\delta_{ut}} \right\rceil$.

Le calcul des pires durées d'exécution réalisé ci-dessus est difficilement compatible avec une application de contrôle de procédé pour laquelle, les durées des tâches en cycles CPU se révèlent le plus souvent très en deçà de la durée d'une unité de temps. Pour le contrôle du pendule, la tâche d'affichage a une durée de 30000 cycles CPU et reste négligeable par rapport à la durée minimum séparant deux "ticks" successifs. La différence d'échelle est telle que l'ensemble des tâches peuvent souvent être toutes exécutées dans une seule unité de temps. Dans notre exemple et selon la méthode décrite ci-dessus, les C_i sont finalement tous unitaires. En outre, plus de 90% du temps CPU est perdu si l'on choisit d'associer un quantum de temps entier à chaque tâche.

Solution retenue : ajout d'un ordonnanceur virtuel. Afin de mieux utiliser la puissance du processeur, nous devons alors modifier ces durées d'exécution, en outre cela permettra de mettre en oeuvre les mécanismes d'ordonnancement classiques. L'idée selon laquelle on pourrait augmenter artificiellement ces durées est mise de côté, puisque cela nuirait à l'asservissement dans la mesure où le temps de réponse de bout en bout du système serait augmenté. Nous écartons également l'utilisation d'une fréquence de "tick" plus grande via les fonctions "tickAnnounce()" et "tickSet()" de VxWorks car (1) WindRiver n'assure plus au delà d'une certaine fréquence la stabilité de son OS et (2), la durée de l'interruption générant le "tick" est du même ordre de grandeur que la durée des tâches courtes (tâches "Aangle", "Butée" et "Alarme"). Nous choisissons donc de réaliser un ordonnanceur au dessus de celui du système d'exploitation afin de gérer plus finement la granularité de notre système. En d'autres termes, nous augmentons virtuellement la fréquence des "ticks".

Afin de réaliser cet ordonnanceur et en utilisant des sémaphores, nous introduisons des points d'arrêts dans les tâches, correspondant aux endroits de préemption éventuelle. Ces points de préemption pourront alors être séparés d'une durée plus petite que celle de l'ordonnanceur de VxWorks. Les points d'arrêts sont en fait obtenus en rajoutant dans le code des tâches des synchronisations avec l'ordonnanceur via l'utilisation de sémaphores.

Ainsi, pour la mise en place d'une granularité plus fine, il y a deux étapes importantes : la réalisation de l'ordonnanceur gérant les différents sémaphores, et l'ajout dans le code des tâches des instructions de prise et de libération de sémaphores.

Fonctionnement de l'ordonnanceur. L'ordonnanceur que nous réalisons au dessus de celui de VxWorks fonctionne suivant ces 5 étapes :

-1- il attend qu'une tâche lui rende la main via une libération de sémaphore,

-2- il fait de l'attente active pour éventuellement finir l'unité de temps débutée par la tâche,

-3- il recherche la tâche suivante, et détermine pendant combien de "ticks" virtuels elle doit s'exécuter (certaines tâches peuvent en effet contenir des parties non pré-

emptibles (primitives d'entrées/sorties)),
-4- il vend un sémaphore pour activer la tâche,
-5- et il se remet enfin en attente sur un sémaphore libéré par la tâche à son prochain point de préemption.

Ces différentes étapes sont ensuite codées en C, et il faut alors déterminer le temps d'exécution de l'ordonnanceur. Dans celui-ci, on ne doit pas prendre en compte l'étape "attente active", puisque par construction, elle est considérée comme faisant partie de la tâche. D'un point de vue pratique, nous décidons d'une part de réaliser un ordonnanceur dont l'exécution n'excède pas 10% du temps attribué à une unité de temps, et d'autre part nous introduisons une nouvelle tâche temps réel qui doit intégrer les "ticks" du noyau VxWorks, toujours générés à intervalle de temps régulier. Pour que ces derniers soient parfaitement synchronisés avec l'ordonnanceur, cette tâche doit être la plus prioritaire possible lors de son activation, on peut ainsi poser : $D_i = C_i$, et on peut faire correspondre sa période T_i avec la durée séparant deux "ticks" successifs. En implémentant l'ordonnanceur, nous avons déterminé qu'il prenait 350 cycles CPU entre l'exécution de deux tâches. Sachant que nous décidons d'une occupation de 10% maximum de l'unité de temps par l'ordonnanceur, on déduit que l'unité de temps à retenir correspond à environ 3500 cycles. Cette imprécision dans la détermination du quantum de temps est dans un premier temps nécessaire, puisqu'il serait maladroit de choisir par exemple un quantum de cette durée si une portion de code ininterruptible d'une tâche correspondait dans le même temps à une durée de 3600 cycles CPU. Avec les valeurs des durées d'exécution des tâches, la durée finalement arrêtée pour l'unité de temps est de 5000 cycles CPU, soit $10\mu s$ avec le processeur utilisé, ce qui correspond à une durée 100 fois plus petite que la durée d'un quantum de temps de VxWorks.

Modification du code des tâches. Une fois la durée du quantum de temps choisie, nous devons rajouter en certains endroits du code de chaque tâche des instructions de prise et de libération des sémaphores d'ordonnancement. Ces nouvelles instructions devraient normalement être séparées d'au plus un quantum de temps. Il se peut cependant qu'un bloc ininterruptible d'instructions ne vienne perturber ce schéma, et dans ce cas, la durée séparant ces deux instructions est supérieure au quantum de temps. Par exemple, la tâche "Potentiomètre" a une durée d'environ 9500 cycles, et possède un bloc ininterruptible de 5 instructions de 600 cycles CPU (cf. figure 5-(a)).

Ce caractère ininterruptible provient le plus souvent des spécificités des routines d'entrées/sorties. Une solution au découpage de la tâche en quantum de temps pourrait alors se traduire par une durée C_i de 3 quanta (cf. figure 5-(b)), cependant, afin d'améliorer l'utilisation processeur, nous préférons choisir une durée de 2 quanta, en imposant une exécution sans préemption de la tâche (cf. figure 5-(c)). On procède ensuite de manière similaire pour les autres tâches de l'application. Les durées d'exécution retenues sont alors reportées dans le tableau 2.

Remarque *:* La durée du quantum de temps peut parfois être plus importante que celle du bloc d'instructions qu'on inclut. Dans ce cas, l'ordonnanceur termine

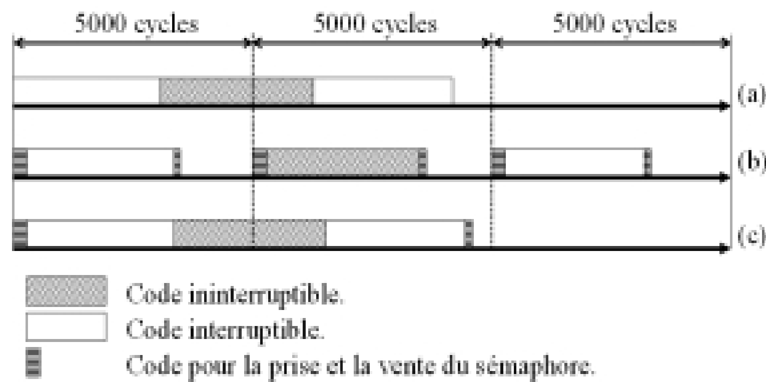

FIG. 5 – Introduction dans le code de la tâche des primitives de libération et de prise du sémaphore. (a) Avant Découpage. (b) Découpage en 3 quanta. (c) Découpage en 2 quanta.

le quantum en effectuant de l'attente active. Cette méthode, bien que n'optimisant en rien l'utilisation de la ressource processeur, a l'avantage de conserver la régularité nécessaire à toutes les applications temps-réel critique. En outre, l'utilisation de sémaphores pour les points d'arrêt peut se révéler dangereuse. Si en effet, pour une raison ou pour une autre, une tâche se bloque, elle ne libérera jamais le sémaphore qui indique la fin de son exécution correspondant au quantum de temps attribué, et l'ordonnanceur ne pourra donc pas prendre ce même sémaphore. Le système sera alors totalement bloqué. Il est ainsi fortement conseillé de conserver l'ordonnanceur de VxWorks, et ce malgré l'ajout d'un ordonnanceur virtuel, car il permet de surveiller les sémaphores par l'intermédiaire de "watchdog". En cas de blocage, une tâche d'arrêt d'urgence du procédé peut alors être lancée.

3.1.2 Les dates de première activation, les délais critiques et les périodes

Nous considérons pour l'application, des tâches à départ simultané, ce qui revient à poser pour chaque date de première activation, $r_{i,1} = 0$. Cela permet en outre de diminuer la durée d'étude de l'ordonnancement : $ppcm\{T_i\}$ au lieu de $t_c + ppcm\{T_i\} + 1$, avec t_c la date du dernier temps creux acyclique [GCG00]. Soulignons que cette hypothèse peut être levée par la suite, dans le but par exemple, d'utiliser des techniques de minimisation de la gigue [DCN01], ou encore afin de gérer d'éventuelles relations de précédences.

Nous considérons aussi que toutes les tâches sont à échéance sur requête, ce qui se traduit pour les paramètres temporels par : $T_i = D_i$. Les périodes des tâches sont alors déterminées en exprimant les contraintes temporelles retenues dans l'espace des temps continu, rapportées à l'espace des temps discrétisé choisi. En outre, on doit éviter de faire une sur-acquisition sur un capteur dont la capacité d'acquisition est par essence limité. Enfin, dans le choix des périodes, on veillera à minimiser la durée d'étude de l'ordonnancement. Deux contraintes de bout en bout sont distinguées pour le pilotage du procédé. La première est liée au temps maximum donné au

système pour arrêter le moteur si une butée est franchie à pleine vitesse par le chariot, cette durée ne doit pas dépasser $10ms$. La seconde concerne la chute du pendule à partir de la verticale : si le temps de réponse du système est supérieure à $5ms$, nous avons alors déterminé que le système ne pouvait plus rattraper le chariot.

Rappelons deux points essentiels dans la détection d'un événement par une tâche : si un événement apparaît juste après l'exécution d'une tâche qu'on suppose intervenir à l'activation, et si la fin de son traitement n'intervient qu'à la fin de l'échéance suivante, on considère alors que la détection, et son traitement sont intervenus au pire après une durée de $D_i + T_i$ unités de temps. Si par contre, comme dans le cas de la tâche "butée" et de la tâche "Alarme", il y a une synchronisation, la détection pour la tâche qui suit se fera avec une durée au pire de D_i. En notant respectivement par T_b et T_a, les périodes de ces deux tâches, et on notant T_m la période de la tâche moteur, la première contrainte temporelle se traduit alors par :

$$2T_b + T_a + 2T_m \leq 10ms. \tag{3.1}$$

En outre, du fait de la synchronisation, on a la relation : $T_b = T_a$. En notant respectivement T_θ, T_p et T_P, les périodes de la tâche d'acquisition de l'angle, de la tâche potentiomètre, et de la tâche PID, l'expression de la seconde contrainte temporelle est donnée par :

$$2T_\theta + 2T_p + 2T_P + T_m \leq 5ms, \tag{3.2}$$

avec toujours la relation : $T_P = T_m$. Ces deux conditions (3.1 et 3.2) forment les principales contraintes temporelles déduites du procédé dans le cadre de ce projet. Il reste cependant des contraintes basées sur la régularité d'exécution que nous n'avons pas encore explorées, et qui feront l'objet d'une prochaine étude.

En procédant par étapes successives, nous proposons finalement la configuration de tâches reportée dans le tableau 2, et qui présente l'ensemble des paramètres temporels de l'application temps réel.

Tâches	r_i	C_i	D_i	T_i	parties ininterruptibles	précédence
butée	0	1	30	30	aucune	$<$ alarme
alarme	0	1	30	30	aucune	$>$ butée
potentiomètre	0	2	30	30	totalement	aucune
angle	0	1	30	30	aucune	aucune
moteur	0	1	10	10	aucune	$>$ calcul_PID
calcul_PID	0	3	10	10	les 2 premières unités	$<$moteur
affichage	0	7	66	66	les 6 dernières unités	aucune
fictive	0	4	4	110	aucune	aucune

TAB. 2 – Paramètres temporels des tâches.

3.2 Technique d'ordonnancement

L'ordonnancement hors-ligne de l'application a été effectué grâce au logiciel PeNSMARTS (Petri Net Scheduling, Modeling and Analysis of Real-Time Systems) [Gro99, CGGC00]. Ce logiciel se base sur une technique énumérative des séquences d'ordonnancements valides, afin de proposer au concepteur d'une application temps réel un choix de séquences d'ordonnancement optimales au vu de certains critères qualitatifs (temps de réponses optimaux pour certaines tâches, etc.).

Cette technique, utilisant une modélisation du système de tâches par réseau de Petri, permet de prendre en compte des systèmes de tâches périodiques complexes (contraintes de précédences, exclusions mutuelles, parties non préemptibles, tâches différées).

Après avoir donné le système de tâches en entrée à PeNSMARTS, l'outil calcule en 35 secondes sur un PENTIUM 450 le graphe contenant tous les ordonnancements valides du système. Ce graphe, constitué de 48392 noeuds, contient environ $1,9.10^{100}$ séquences d'ordonnancement valides. Etant donné que les tâches “potentiomètre” et “angle” sont des tâches d'acquisition desquelles dépend le PID, nous les rendons régulières en leur imposant les temps de réponse respectifs 6 et 20 unités de temps. Environ 10^{65} séquences respectent cette contrainte. Afin d'optimiser le calcul du PID et son application au moteur, nous cherchons alors à minimiser le temps de réponse moyen des tâches $calcul_PID$ et $moteur$. Il reste alors environ $1,8.10^{20}$ séquences. Parmi celles-ci, toutes les séquences répondent aux critères choisis : régularité des tâches d'acquisition du potentiomètre et de l'angle, et temps de réponse moyen minimal pour les tâches $calcul_PID$ et $moteur$.

Il reste alors à intégrer dans le programme de contrôle la table contenant la séquence choisie fournie en langage C par PeNSMARTS.

4 Conclusion

Nous avons présenté à travers un exemple l'utilisation d'une plate-forme de test d'outils de validation de systèmes temps réel. L'exemple présenté suit le cycle de développement typique d'une application temps réel afin de démontrer l'applicabilité de techniques de validation à des exemples concrets.

La principale difficulté résidait dans le fait que l'ordonnanceur fourni avec des noyaux temps réel du commerce possède un quantum de temps tellement grand comparé à la puissance des processeurs actuels, qu'il est difficile de mener une démarche de validation basée sur l'ordonnancement. Un ordonnanceur virtuel a alors été mis en oeuvre, afin de diminuer le quantum de temps ordonnanceur, et rendre attractives les techniques d'ordonnancement.

L'avantage de cette méthode est la diminution drastique du temps de réponse de bout en bout du système. En effet, en utilisant l'ordonnanceur de base du système, toutes les tâches auraient été de durée unitaire, et d'une période égale au nombre de tâches utilisées. Ainsi, chaque tâche aurait eu une période de 7 millisecondes. L'ajout d'un ordonnanceur virtuel donne à la tâche la plus fréquente une période de

$100\mu s$ et à la moins fréquente une période de $1,1ms$. De plus, nous avons souligné qu'une telle granularité ne pouvait être atteinte en utilisant les "ticks" gérés par le système d'exploitation, puisque la durée de traitement associé représentait la durée des tâches courtes de notre application.

Références

[Aud91] N.C. Audsley, *Optimal priority assignment and feasibility of static priority tasks with arbitrary start times*, Tech. Report YCS-164, University of York, nov. 1991.

[Bak91] T.P. Baker, *Stack-based scheduling of real-time processes*, Real-Time Systems **3** (1991), 67–99.

[BHR90] S.K. Baruah, R.R. Howell, and L.E. Rosier, *Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor*, Real-Time Systems **2** (1990), 301–324.

[CGGC00] A. Choquet-Geniet, E. Grolleau, and F. Cottet, *Etude hors-ligne d'une application temps réel à contraintes strictes*, Technique et Science Informatiques **19** (2000), no. 10, 1373–1398.

[CL90] M. Chen and K. Lin, *Dynamic priority ceilings : a concurrency protocol for real-time systems*, Real-Time Systems **2(4)** (1990), 325–346.

[CNR88] CNRS : Groupe de réflexion temps-réel du CNRS, *Le temps-réel*, Technique et Science Informatiques - TSI **7** (1988), no. 5, 493–500.

[DCN01] L. David, F. Cottet, and N. Nissanke, *Jitter control in on-line scheduling of dependent real-time tasks*, Proc. of the 22nd IEEE Real-Time Systems Symposium, RTSS'01 (London, UK), December 2001, pp. 49–58.

[Der74] M.L. Dertouzos, *Control robotics : the procedural control of physical processors*, Proc. of IFIP Congress, 1974, pp. 807–813.

[DM89] M.L. Dertouzos and A.K. Mok, *Multiprocessor on-line scheduling of hard real-time tasks*, IEEE Transactions on Software Engineering **15(12)** (Dec. 1989), 1497–1506.

[Gal95] B.O. Gallmeister, *Posix.4 : Programming for the real world*, O'Reilly and Associates, 1995.

[GCG00] E. Grolleau and A. Choquet-Geniet, *Cyclicité des ordonnancements de systèmes de tâches périodiques différées.*, Proc. of RTS'2000 (Paris, France), 2000, pp. 216–228.

[Gom84] Hassan Gomaa, *A software design method for real-time systems*, Communications of the ACM **27** (1984), no. 9, 938–949.

[Gro99] E. Grolleau, *Ordonnancement temps-réel hors-ligne optimal à l'aide de réseaux de pétri en environnement monoprocesseur et multiprocesseur*, Ph.D. thesis, ENSMA - Université de Poitiers, nov. 1999.

[Lab74] J. Labetoulle, *Un algorihtme optimal pour la gestion des processus en temps réel*, Revue Française d'Automatique, Informatique et Recherche Opérationnelle (Fév. 1974), 11–17.

[LL73] C.L. Liu and J.W. Layland, *Scheduling algorithms for mutltiprogramming in real-time environnement*, Journal of the ACM **20(1)** (1973), 46–61.

[LW82] J. Leung and J. Whitehead, *On the complexity of fixed-priority scheduling of periodic, real-time tasks*, Performance Evaluation (Netherland) **2(4)** (1982), p. 237–250.

[Mok83] A.K. Mok, *Fundamental design problems for the hard real-time environments*, Ph.D. thesis, MIT, May 1983.

[SRL90] L. Sha, R. Rajkumar, and J.P. Lehoczky, *Priority inheritance protocols : an approach to real-time synchronisation*, IEEE Transactions on Computers **39(9)** (1990), 1175–1185.

[Sys99] WindRiver Systems, *Vxworks benchmark methodology report*, Tech. report, WindRiver Systems, 1999.

[Tör98] M. Törngren, *Fundamentals of implementing real-time control applications in distributed computer systems*, Real-Time Systems **14** (1998), 219–250.

[WM86] Paul T. Ward and Stephen J. Mellor, *Structured development for real-time systems*, vol. volume 3 : Implementation Modeling Techniques, Yourdon, Inc., Englewood Cliffs, New Jersey, 1986.

[XP92] J. Xu and D.L. Parnas, *Pre-run-time scheduling of processes with exclusion relations on nested or overlapping critical sections*, Phoenix Conference on Computers and Communications (Phoenix, USA), Apr. 1992, pp. 6471–6479.