INTEGRATION D'UN CONTROLE DE CHARGE PAR IMPORTANCE AU SEIN DU SYSTEME RT-LINUX

Joëlle Delacroix, Christophe Ménival
Conservatoire National des Arts et Métiers
292 rue Saint Martin
75141 Paris Cedex 03

Tel : 01 40 27 28 81
Fax : 01 40 27 27 02
delacroix@cnam.fr

Résumé

Notre travail s'intéresse aux applications temps réel comportant à la fois des tâches à contraintes strictes et des tâches à contraintes relatives et développe une méthode de prise en compte des surcharges permettant de minimiser l'occurrence de fautes temporelles ainsi que leurs conséquences sur le comportement de l'application : cette méthode s'appuie sur la notion de contrôleur de charge et sur un modèle de tâches multi-modes dédié à la prise en compte des conséquences des fautes temporelles et des suppressions d'exécutions. Nous présentons plus particulièrement l'implémentation de cette méthode au sein du système RT-Linux[1]

Mots-Clés : surcharge, contrôleur de charge, modèle de tâches, système RT-Linux

1. Introduction

Les applications temps réel sont généralement classées selon deux groupes extrêmes, en fonction du type de contraintes temporelles auxquelles sont soumises les tâches qui les composent : ainsi on parle d'une part de systèmes temps réel à contraintes strictes pour lesquels aucune faute temporelle ne peut être tolérée sous peine de graves conséquences pour le système lui-même et son environnement et d'autre part de systèmes temps réel à contraintes relatives pour lesquels l'occurrence de fautes temporelles est possible sans engendrer d'autres conséquences qu'une perte de performance. La réalité se situe pourtant souvent entre ces deux extrêmes : une application temps réel est ainsi composée d'un ensemble de tâches à contraintes strictes pour lesquelles aucune faute temporelle ne peut être admise et d'un ensemble de tâches à contraintes relatives pour lesquelles le dépassement de certaines échéances est autorisée. Notre travail s'intéresse à ces applications construites sur ces deux types de contraintes et développe une méthode de prise en compte des surcharges permettant de minimiser l'occurrence de fautes temporelles ainsi que leurs conséquences sur le comportement de l'application : cette méthode s'appuie sur la notion de contrôleur de charge et sur un modèle de tâches multi-modes dédié à la prise en compte des conséquences des fautes temporelles et des suppressions d'exécutions [Delacroix 98].

Les applications que nous considérons sont des applications temps réel multitâches pour lesquelles les tâches sont caractérisées par leur urgence et par leur importance dans l'application et sont ordonnancées de manière dynamique par un algorithme tel que "Earliest Deadline" [Liu 73]. Nous considérons que les tâches ne partagent pas d'autres ressources que le processeur mais peuvent cependant être liées par des contraintes de précédence. Les tâches sont soit des tâches périodiques se réveillant à intervalles réguliers liées à la supervision courante du procédé contrôlé, soit des tâches apériodiques

[1] Toute information sur RT-Linux est disponible à l'URL suivante : http://rtlinux.cs.nmt.edu

se réveillant de manière asynchrone, suite à un événement de type alarme. L'occurrence de ces tâches apériodiques, ainsi que les variations des temps d'exécution des tâches, peuvent placer le système temps réel en situation de surcharge, situation dans laquelle des tâches de l'application vont commettre des fautes temporelles. Dans cette situation de surcharge, l'ordonnancement doit s'effectuer de manière à assurer une exécution de l'application permettant de maintenir le procédé contrôlé dans un état sécuritaire : pour cela, nous choisissons au moment d'une surcharge, de favoriser l'exécution des tâches les plus importantes de l'application et d'effectuer des suppressions d'exécutions des tâches les moins importantes. Les suppressions d'exécutions sont réalisées par un contrôleur de charge placé en amont de l'ordonnanceur. Notre modèle multi-modes permet de supporter les conséquences de ces suppressions.

Dans ce cadre, notre papier présente l'implémentation faite de ce contrôleur de charge et du modèle de tâches multi-modes au sein du système Real-time-Linux ou RT-Linux[2] [Epplin 97][Barabanov 96]. RT-Linux est une extension du système d'exploitation Linux permettant l'exécution de tâches soumises à des contraintes de temps. Dans cette extension, le système Linux classique a été modifié de manière à former un noyau de base capable de partager le contrôle du système avec un noyau temps réel. Le système résultant peut être vu comme un noyau double dans lequel le noyau temps réel a la plus haute priorité : ainsi, le noyau Linux classique est vu par le noyau temps réel comme étant la tâche de plus basse priorité parmi les tâches temps réel (figure 1).

La structure du système RT-Linux et Linux est similaire à celle des Unix classiques : c'est une architecture monolitique incluant les différents services systèmes tels que le système de gestion de fichiers, les pilotes de périphériques, etc…, liés de manière statique dans l'image noyau chargée au moment de l'amorçage du système (boot). La notion de modules chargeables [Welsh 95] permet cependant de développer de manière séparée à l'image noyau statique des services supplémentaires et de les charger dynamiquement au sein du système (figure 1). Ainsi l'ordonnanceur du noyau RT-Linux ainsi que les applications temps réel développées pour RT-Linux se présentent comme des modules chargeables vis-à-vis du noyau. Nous avons nous-mêmes exploité cette notion de module chargeable pour intégrer notre contrôleur de charge.

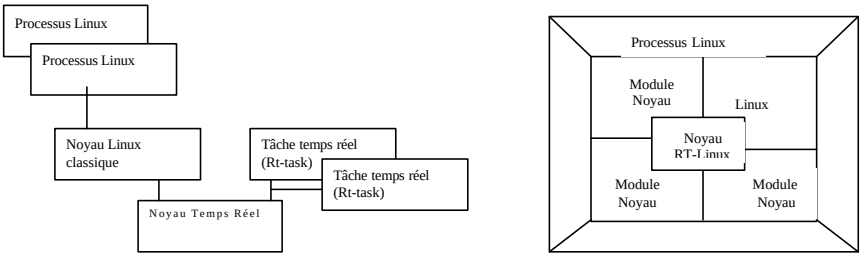

Figure 1 : structure de RT-Linux en terme de tâches et de modules

Nous présentons tout d'abord l'ordonnanceur et le modèle d'application temps réel nâtifs de RT-Linux. Puis, nous décrivons notre modèle de tâches multi-modes et le fonctionnement du contrôleur de charge. Ceci fait, nous présentons l'implémentation que nous en avons faite au sein de RT-Linux.

[2] Cette implémentation s'effectue dans le cadre du mémoire d'ingénieur Cnam en informatique de C. Ménival

2. Présentation de l'ordonnanceur natif de RT-Linux

L'ordonnanceur de RT-Linux se présente sous la forme d'un module chargeable kernel/rt_prio_sched.c qui définit la structure des tâches temps réel, les appels système permettant la gestion de ces tâches et la politique d'ordonnancement.

2.1 Structure et diagrammes d'états des tâches

Une application temps réel composée de tâches temps réel (RT_TASK) est également définie comme un module chargeable. Les tâches temps réel, qui peuvent être soit des tâches périodiques, soit des tâches apériodiques, s'exécutent dans l'espace d'adressage du noyau, afin de limiter les surcoûts liés aux opérations de commutation. Chaque tâche est décrite par une structure RT_TASK (fichier linux/rt_sched.h) regroupant des informations telles que l'état de la tâche, sa priorité, sa période, etc... (figure 2). Les structures RT_TASK sont organisées comme une liste simplement chainée, terminée par une tâche particulière, la tâche rt_linux_task, qui représente au sein du linux temps réel, le système linux classique.

```
struct rt_task_struct {
        int *stack;
        int uses_fp;
        int magic;
        int state;              /* état de la tâche */
        int *stack_bottom;
        int priority;           /* priorité de la tâche */
        RTIME period;/* période de la tâche */
        RTIME resume_time;      /* date de réveil */
        struct rt_task_struct *next;
        RTL_FPU_CONTEXT fpu_regs; };

typedef struct rt_task_struct RT_TASK;
```

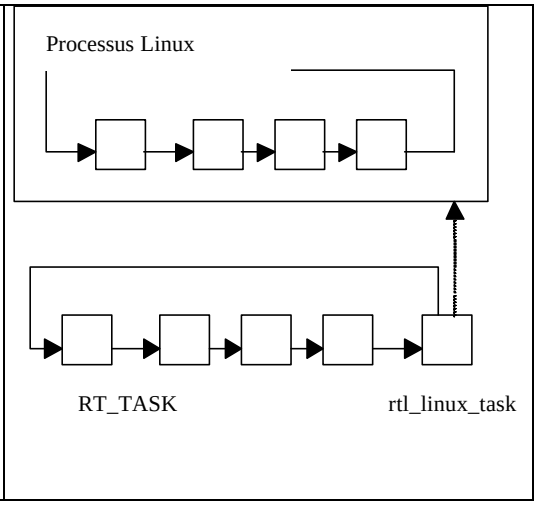

Figure 2 : structure d'une tâche temps réel (RT_TASK)

2.1.1. Diagrammes d'états des tâches périodiques et apériodiques

La figure 3 définit le diagramme d'états d'une tâche périodique et d'une tâche apériodique au sein de RT_Linux.

La primitive *rt_task_init()* crée une nouvelle tâche et met celle-ci dans l'état Crée-dormante (*dormant*), état pour lequel la tâche existe mais n'est pas connue de l'ordonnanceur. Une tâche est détruite par un appel à la primitive *rt_task_delete()*.

La prise en compte d'une nouvelle tâche par l'ordonnanceur se fait de manière différente selon si la tâche est périodique ou apériodique. Dans le cas d'une tâche périodique, cette prise en compte se fait par l'intermédiaire de la primitive *rt_task_make_periodic()* qui initialise la date de premier réveil de la tâche et sa période. La tâche passe dans l'état Cree_avec_délai (*delayed*) et passe dans l'état Prête (*ready*) lorsque sa date de réveil est atteinte. Lorsque la tâche a terminé son exécution, elle se suspend jusqu'à sa prochaine période par un appel à la primitive *rt_task_wait()*. Dans le cas d'une tâche apériodique, la prise en compte par l'ordonnanceur d'une nouvelle tâche se fait par l'intermédiaire de la levée d'un sous-traitant d'événements (*handler*) associé à

l'événement de réveil de la tâche apériodique. Ce sous-traitant d'événements, attaché à l'événement de réveil par le biais de la primitive *request_RTirq()*(défini dans /*linux/arch/i386/kernel/irq.c*), contient un appel à la primitive *rt_task_wakeup()* qui place la tâche apériodique dans l'état prêt. La tâche apériodique, une fois son code exécuté, fait un appel à la primitive *rt_task_suspend()* ce qui la suspend jusqu'à son prochain réveil. Cette primitive peut être également utilisée pour suspendre pendant un certain intervalle de temps l'exécution d'une tâche périodique.

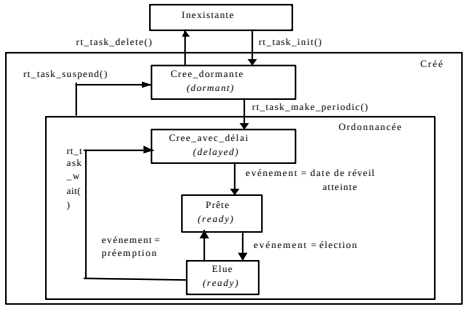

diagramme d'états pour une tâche périodique

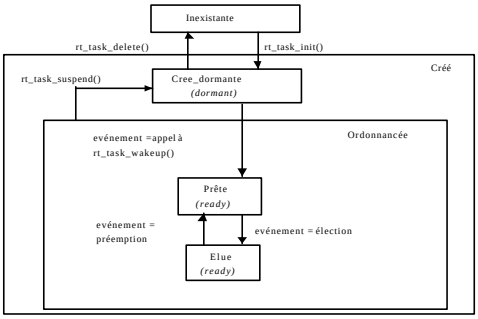

diagramme d'états pour une tâche apériodique

Figure 3

La figure 4 ci-dessous présente une application temps réel RT-Linux composée d'une tâche périodique et d'une tâche apériodique.

```
#define MODULE
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/version.h>
#include <linux/errno.h>
#include <linux/rt_sched.h>
#include <linux/arch/i386/kernel/irq.h>

RT_TASK tasks[2];

void f_periodique_(int t) {          /* la fonction exécutée par la tâche périodique 1 */
        while (1) {
                        quelque_chose....
                        rt_task_wait(); }}

void f_aperiodique (int t) {         /* la fonction exécutée par la tâche apériodique  */
        quelque_chose....
        rt_task_suspend(&task([1]);  }

int ap_handler() {   /* le sous-traitant d'événements associé à l'IRQ 2 qui réveille la tâche apériodique */
        rt_task_wakeup(&task([1]); }

int init_module(void) {
        rt_task_init(&tasks[0], f_periodique, 0, 3000, 4); /* création de la tâche périodique */
```

```
        rt_task_init(&tasks[1], f_aperiodique, 1, 3000, 5); /* création de la tâche apériodique */
        rt_task_make_periodic((&task[0], 5, 10); /* initialisation de la tâche périodique */
        request_RTirq(2, &ap_handler); /* attachement du handler à l'IRQ 2 */
        return 0; }

void cleanup_module(void)
{
        rt_task_delete(&tasks[0]); /* destruction de la tâche périodique */
        rt_task_delete(&tasks[1]); /* destruction de la tâche apériodique */
        free_RTirq(2);  /* liberation de l'IRQ 2 */
}
```

Figure 4 : programmation de tâches périodiques et apériodiques

2.2 Type d'ordonnanceur

L'ordonnanceur intégré à RT-Linux (fonction *rtl_schedule()*) est un ordonnanceur préemptif à priorité fixe. A tout instant, la tâche temps réel de plus grande priorité est la tâche élue. Si aucune tâche temps réel n'est prête, la tâche Linux (*rtl_linux_task*) correspondant à l'exécution du système Linux classique est élue par l'ordonnanceur RT-Linux.

3. Pourquoi modifier l'ordonnanceur natif ?

3.1. Introduction

L'ordonnanceur intégré à RT-Linux est donc un ordonnanceur préemptif à priorité fixe. Bien que ce type d'ordonnanceur soit celui couramment implémenté dans les noyaux dits temps-réel, il s'avère mal adapté à la gestion et au respect des contraintes temporelles associées aux tâches temps réel, ceci pour deux raisons essentielles :

- la priorité attachée à chaque tâche sur laquelle se base la politique d'ordonnancement n'est pas un bon indicateur de l'urgence d'une tâche,
- il n'existe pas de test d'acceptabilité de configurations de tâches périodiques permettant de certifier l'ordonnancement réalisé.

Une bonne politique d'ordonnancement temps réel est une politique pour laquelle le facteur temps intervient directement dans les prises de décisions de l'ordonnanceur : pour cela, il faut que la priorité de chaque tâche soit le reflet de son urgence. Parmi toutes les politiques d'ordonnancement existantes, la politique préemptive "Earliest Deadline" qui élit à tout instant t, la tâche de plus proche échéance, est celle qui répond le mieux à l'exigence énoncée.

Malheureusement, la politique "Earliest Deadline" est instable en période de surcharge : lorsque des fautes temporelles surviennent, soit suite à des réveils de tâches apériodiques, soit suite à des temps d'exécutions variables, la politique "Earliest Deadline" ne permet pas de contrôler l'identité des tâches touchées par les fautes temporelles. Souvent, une faute temporelle en provoque d'autres, créant alors un phénomène d'avalanche de fautes temporelles.

Pour utiliser de manière satisfaisante la politique "Earliest Deadline" afin d'ordonnancer une application temps réel soumise à des fluctuations de temps d'exécutions ou à des exécutions de tâches apériodiques, il faut adjoindre à celle-ci un mécanisme de contrôle de charge par importance qui va permettre de détecter l'occurrence de surcharge

et de contrôler l'identité des tâches dont on veut favoriser les exécutions. Ce mécanisme de contrôle de charge résorbe une surcharge en effectuant des suppressions d'exécutions au sein de l'ensemble des tâches actives [Delacroix 96].

Le modèle à quatre paramètres (date de réveil, temps d'exécution maximal, délai critique et période) classiquement utilisé pour décrire les tâches d'une application temps réel doit être enrichi de plusieurs manières en vue du contrôle de charge [Delacroix 98]:

- un paramètre supplémentaire doit être introduit de manière à qualifier la primordialité d'une tâche dans l'application séparément de son urgence : c'est le rôle du paramètre importance. Le contrôleur de charge choisit les exécutions de tâches à supprimer en fonction de cette importance.
- le comportement de la tâche doit être amélioré de manière à permettre la prise en compte des conséquences des suppressions d'exécutions au niveau de la tâche elle-même et au niveau des tâches dépendant de l'exécution supprimée : pour cela nous avons défini des propriétés de gestion d'exécutions qui spécifient de quelle manière une exécution de tâche peut être supprimée et nous avons spécifié un nouveau modèle de tâches multi-modes. Une tâche est alors composée de plusieurs modes, un mode normal dont l'exécution correspond à l'exécution courante de la tâche, et des modes de survies qui sont activés lorsque l'exécution du mode normal de la tâche est arrêtée par le contrôle de charge. Ces modes de survies contiennent par exemple des actions de traitements palliatives ou des ordres de suppressions d'exécutions à destination d'autres tâches de l'application.

Nous présentons maintenant notre nouveau modèle de tâches, puis la politique de contrôle de charge associée.

3.2. Un modèle de tâches adapté aux suppressions d'exécutions

La figure 5 décrit le squelette d'une tâche établie avec le modèle multi-modes. Une tâche est composée de quatre modes et comporte une entrée pour chaque mode qui contient les actions à exécuter lorsque le mode est activé ainsi qu'une description des paramètres temporels et qualitatifs du mode. La tâche T_i est alors décrite par l'ensemble de ses modes $\{M_{ni}, M_{aji}, M_{rvi}, M_{fti}\}$, où M_{ni} est le mode normal de T_i, M_{aji} est le mode ajournement de T_i, M_{rvi} est le mode révocation de T_i and M_{fti} est le mode faute temporelle de $T_{i.}$

Le mode normal est le mode dont l'exécution est demandée lorsqu'une tâche devient active et correspond à l'exécution normale de la tâche. Il est caractérisé par un temps d'exécution maximal C_n, une échéance d_n, une importance Imp_n, et un couple de propriétés dites propriétés de gestion d'exécution qui définissent la manière dont l'exécution du mode normal peut être supprimée par le contrôle de charge. Nous distinguons deux propriétés de gestion d'exécution, l'ajournabilité et la révocabilité. L'exécution d'un mode normal est dite ajournable si celle-ci peut être supprimée avant qu'elle n'ait commencé : nous disons alors que le contrôle de charge ajourne le mode. L'exécution d'un mode normal est dite révocable si celle-ci peut être supprimée alors qu'elle est déjà commencée. Nous disons alors que le contrôle de charge révoque le mode. Ainsi un mode normal peut être :

- ajournable et révocable : l'exécution du mode est optionnelle et peut être supprimée à tout moment.
- ajournable mais non-révocable : une fois commencée, l'exécution du mode doit être complète.
- non ajournable et révocable : l'exécution du mode ne peut être supprimée tant que celle-ci n'a pas commencée.

- non ajournable et non révocable : l'exécution complète du mode est obligatoire

Les modes de survie "ajournement" et "révocation" contiennent les actions qui doivent être exécutées lorsque l'exécution du mode normal a été respectivement ajournée ou révoquée par le contrôle de charge. Ces actions permettent par exemple de communiquer des résultats partiels aux autres tâches de l'application, d'exécuter un travail palliatif ou encore constituent des demandes de suppressions d'exécutions d'autres tâches de l'application liées à la tâche supprimée par des liens de précédence. Chacun de ces deux modes de survie n'est défini que si le mode normal possède les propriétés d'ajournabilité et de révocabilité et est caractérisé par un temps d'exécution maximal $C_{aj/rv}$ et une échéance $d_{aj/rv}$ égale à celle du mode normal. Leur exécution est obligatoire et ne peut pas être supprimée par le contrôle de charge.

Le mode de survie faute temporelle est exécuté lorsque l'un des modes précédents de la tâche commet une faute temporelle. Ce mode est divisé en trois parties qui correspondent respectivement aux actions à exécuter lorsque la faute temporelle est commise par le mode normal, par le mode ajournement ou par le mode révocation et qui sont seulement caractérisées par un temps d'exécution maximal Cxf, x pour n, aj ou rv. Chacune des trois parties est optionnelle ; aussi si aucune action de reprise n'est définie dans le mode faute temporelle pour l'un des modes fautif, alors ce mode stoppe simplement son exécution. Ce mode de survie faute temporelle est rendu nécessaire par les propriétés de gestion d'exécution qui, selon leur valeur, peuvent interdire la suppression des exécutions de certains modes normaux.

```
Task  T is
debut
Mode Normal :
    Actions du mode normal (Cn, dn, propriétés, Impn)
Mode de survie Ajournement :
    Actions du mode ajournement (Caj, daj, exécution obligatoire)
Mode de survie Révocation :
    Actions du mode révocation (Crv, drv, exécution obligatoire)
Mode de survie Faute Temporelle :
        Actions si faute temporelle du mode normal (Cnmf)
        Actions si faute temporelle du mode ajournement (Cajf)
        Actions si faute temporelle du mode révocation (Crvf)
fin;
```

Figure 5 : modèle multi-modes

3.3. La politique de contrôle par importance

La politique de contrôle de charge se déroule en deux grandes étapes, dont l'exécution est à la charge d'un module spécifique placé en amont de l'ordonnanceur, le contrôleur de charge (figure 6) :

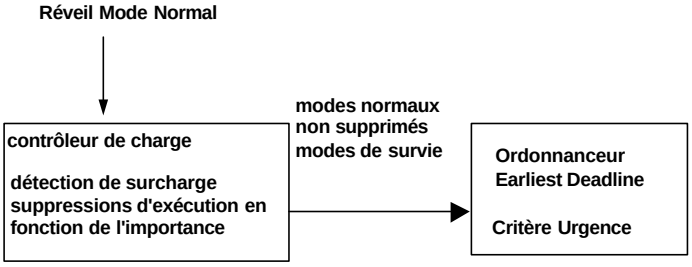

Figure 6 : le contrôleur de charge

- la première étape réalise la détection de la surcharge. A chaque nouvelle demande d'exécution d'une tâche T_i ce qui correspond à l'activation de son mode normal M_{ni}, le contrôleur de charge évalue la laxité courante de chaque mode actif, c'est-à-dire le temps durant lequel le mode peut attendre avant de reprendre son exécution sans qu'il commette de faute temporelle. Si la laxité de tous les modes actifs est positive, il n'y a pas surcharge et la nouvelle tâche est acceptée pour être ordonnancée selon "Earliest Deadline". Au contraire, si un mode possède une laxité négative, alors ce mode est en retard et va commettre une faute temporelle : le processeur est surchargé. Plus précisément, soit M = {M_{xi}(r, C_{xi}(r), d_{xi}, Imp_{xi}) avec x = n ou aj ou av, i = 1 to m} l'ensemble des modes actifs à la date r, triés par échéance croissante. Alors la laxité du mode M_{xi} est définie comme étant égale à : $L_{xi}(r) = d_{xi} - r - \Sigma C_{xk}(r)$, $d_{xk} \leq d_{ik}$. L'ensemble M n'est pas ordonnançable s'il existe un mode M_{xf} pour lequel sa laxité $L_{xf}(r)$ est négative. Il y a surcharge ; le mode M_{xf} est fautif et la valeur de la surcharge est égale à $| L_{xf}(r) |$.

- la seconde étape réalise la résorption de la surcharge. Le contrôleur de charge utilise alors le paramètre importance pour effectuer des suppressions d'exécutions au sein de l'ensemble des modes normaux. Soit M' = {M_{ni}(r, C_{ni}(r), d_{ni}, Imp_{ni}), i = 1 to p}, M' $\subseteq$ M, l'ensemble des modes normaux qui peuvent être soit ajournés, soit révoqués, dont l'échéance d_{ni} est inférieure ou égale à d_{xf}. Pour résorber la surcharge, le contrôleur de charge supprime les exécutions de modes normaux, un à un, par ordre croissant d'importance. Pour chaque suppression effectuée, le contrôleur de charge active le mode de survie correspondant (activation du mode ajournement si le mode normal est ajourné, activation du mode révocation si le mode normal est révoqué). La surcharge est résorbée lorsque la somme des temps d'exécution des modes normaux supprimés plus la somme des temps d'exécution des modes de survie activés est supérieure ou égale à la valeur de la surcharge. S'il n'existe pas suffisamment de modes normaux dans l'ensemble M', alors la surcharge n'est pas résorbée. Des fautes temporelles subsistent qui seront prises en charge par l'exécution des modes de survie faute temporelle.

4. Intégration de la politique à importance au sein de RT-Linux

Nous intégrons la politique de contrôle de charge sous la forme d'un module chargeable dans l'espace du noyau. Ce module chargeable intégre :

- la définition du modèle de tâches multi-modes. Pour l'instant seuls les modes normaux et les modes de survie Ajournement et Révocation sont implantés.
- un ordonnanceur "Earliest Deadline" et le contrôleur de charge.

4.1. Structure et diagrammes d'états des tâches

La structure et le diagramme d'états des tâches temps réel de RT-Linux sont modifiés de manière à intégrer

- les paramètres temporels relatifs à Earliest Deadline et au contrôle de charge (échéance, temps d'exécution maximal estimé, temps d'exécution restant,

propriétés de gestion d'exécution, paramètre importance)
- la structure multi-modes des tâches

4.1.1. Structure des tâches périodiques et apériodiques

Pour représenter la structure à trois modes d'une tâche temps réel T, nous associons à chaque tâche trois descripteurs de tâche de type RT_TASK distincts, dédiés chacun à un mode de la tâche. Les descripteurs de tâches correspondant aux modes normaux sont chainés entre eux selon une liste simple et chaque descripteur de mode normal référence les descripteurs des modes de survie qui lui sont associés (figure 7).

Un descripteur de tâche est composé des champs initialement définis dans la structure RT_TASK native de RT_Linux auxquels ont été ajoutés (figure 7) :

- les champs *RTIME max_exec_time* et *RTIME rem_exec_time* qui sont respectivement le temps maximal d'exécution de la tâche et son temps d'exécution restant
- les champs *RTIME absolute deadline* et *RTIME relative deadline* qui définissent respectivement l'échéance absolue de la requête courante de la tâche et l'échéance "relative" de la tâche. L'échéance absolue est égale à *resume_time* + *relative_deadline*.
- le champs *int importance* qui code l'importance de la tâche
- le champs *int properties[2]* qui permet de coder les propriétés de gestion d'exécution attachées à la tâche : *properties[0]* code la propriété d'ajournement et prend pour valeur soit *ADJOURNABLE*, soit *UNADJOURNABLE; properties[1]* code la propriété de révocation et prend pour valeur soit *ABORTABLE*, soit *UNABORTABLE*
- les champs *rt_task_struct *next_ready* et *rt_task_struct *prev_ready* qui permettent un chaînage double des tâches prêtes par ordre croissant d'échéance. Deux fonctions sont associées à cette liste chaînée : la fonction *ote_ready(RT_TASK *task)* ôte la tache passée en paramètre de la liste des tâches prêtes tandis que la fonction *insere_ready(RT_TASK *task)* insère la tâche passée en paramètre dans la liste des tâches prêtes et ceci en fonction de son échéance.
- les champs *rt_task_struct *next_imp* et *rt_task_struct *prev_imp* qui permettent un chaînage double des tâches prêtes par ordre croissant d'importance. Deux fonctions sont associées à cette liste chaînée : la fonction *ote_imp(RT_TASK *task)* ôte la tâche passée en paramètre de la liste des tâches par importance tandis que la fonction *insere_imp(RT_TASK *task)* insère la tâche passée en paramètre dans la liste des tâches par importance et ceci en fonction de son importance.
- le champ *type* qui prend trois valeurs – normal, ajournement, révocation – et permet de spécifier à quel mode de la tâche est attaché le descripteur.
- les champs *rt_task_struct *mode_Aj* et *rt_task_struct *mode_Rv* qui référence pour une structure correspondant à un mode normal, les structures correspondant aux modes de survie Ajournement et Révocation.

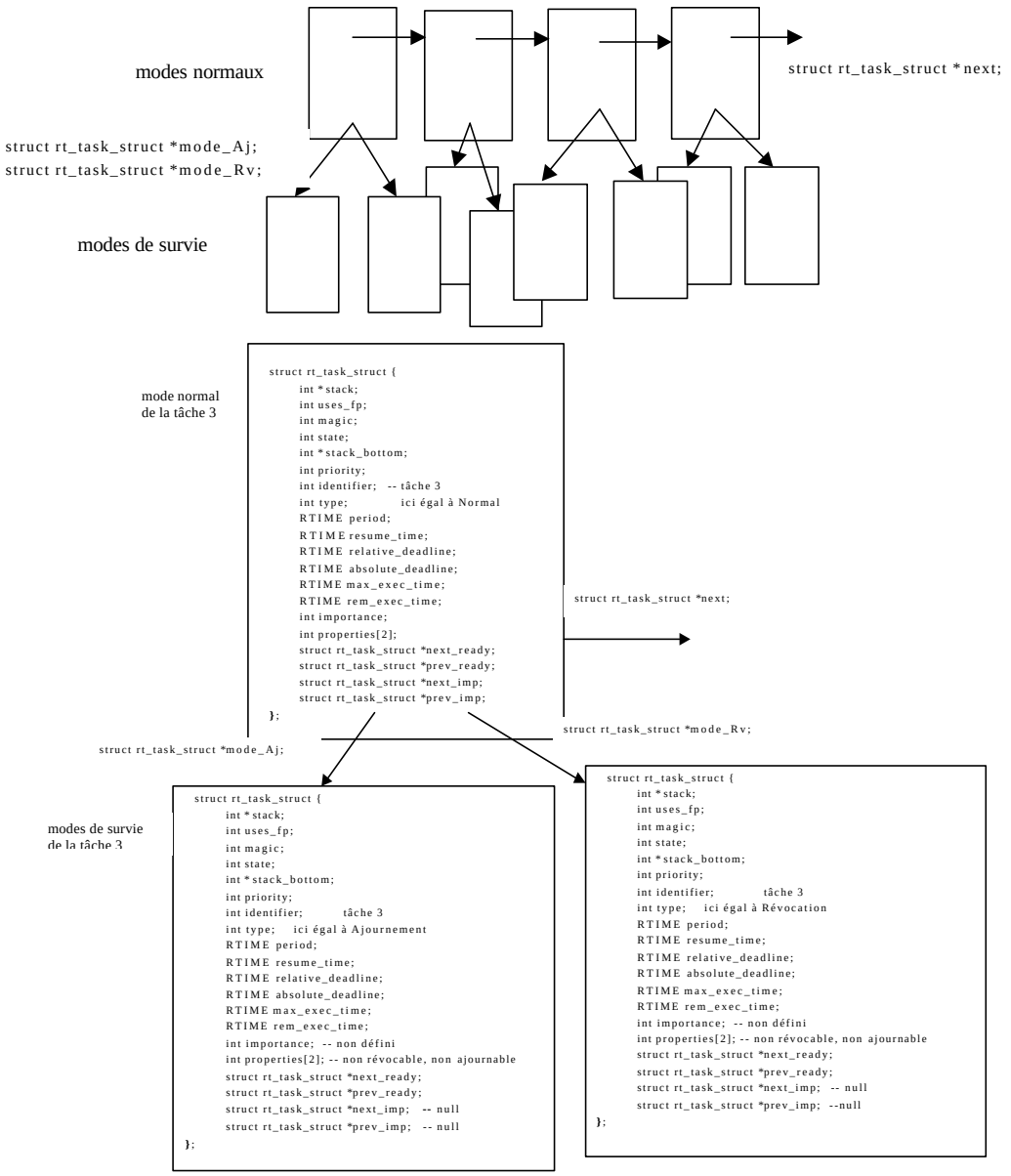

Figure 7 : structure des tâches

4.1.2. Diagrammes d'états des tâches périodiques et apériodiques

La figure 8 donne le diagramme d'états d'une tâche périodique pour le contrôle de charge. Chaque diagramme d'états est lui-même divisé en trois parties, correspondant respectivement au diagramme d'états du mode normal et des deux modes de survie de la tâche (la partie diagramme d'état d'un mode de survie est en deux exemplaires, un par mode de survie). Le diagramme d'états d'une tâche apériodique est similaire à la seule différence du diagramme du mode normal qui doit être remplacé par celui de la figure 3.
Tous les modes de toutes les tâches d'une application sont instanciées au démarrage de celle-ci par la primitive *rt_task_init().*

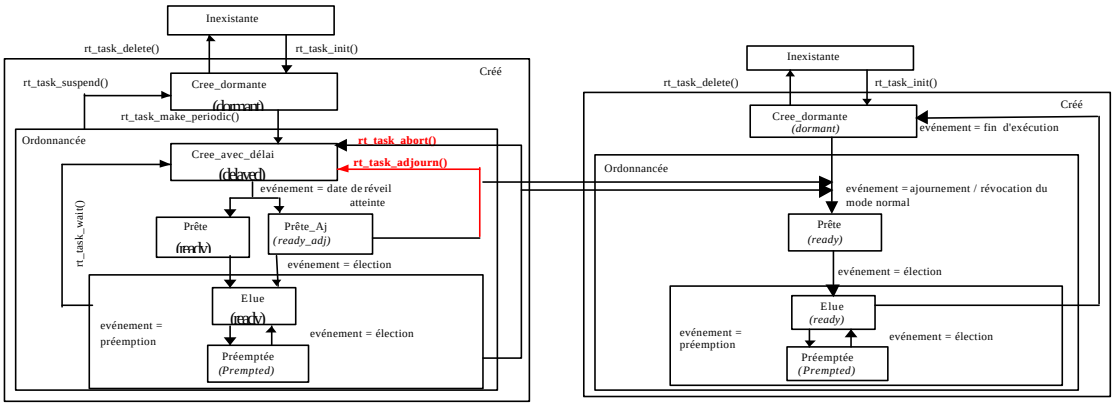

Figure 8 : diagramme d'états pour une tâche périodique

Deux états ont été introduits :

- l'état Prêt_Aj (*Ready_Adj*) permet de différencier les tâches prêtes non ajournables des tâches prêtes ajournables, auxquelles une opération d'ajournement peut être appliquée.
- l'état Préemptée (*Preempted*) qui dédouble l'état prêt et permet de gérer l'opération de révocation. En effet, une tâche ne peut être révoquée qu'une fois son exécution commencée. Or un seul état prêt ne permet pas de faire la distinction entre les tâches prêtes n'ayant pas encore commencées leur exécution (tâches non encore préemptées) des tâches prêtes ayant déjà commencées leur exécution (tâches préemptées)

Deux primitives sont également introduites :

- la primitive d'ajournement de tâche : int rt_task_adjourn (RT_TASK *task) qui fait quitter à une tâche ajournable (état Prêt_Aj) le macro-état "ordonnancée" pour la replacer dans le macro-état "crée mais non ordonnancée" (état Crée_avec_délai pour une tâche périodique et crée_dormant pour une tâche apériodique). Cette primitive par ailleurs active le mode ajournement de la tâche en le mettant dans l'état prêt.
- la primitive de révocation de tâche : int rt_task_abort (RT_TASK *task) qui fait passer une tâche active ou préemptée dans le macro-état "crée mais non ordonnancée" (état Crée_avec_délai pour une tâche périodique et crée_dormant pour une tâche apériodique). Cette primitive par ailleurs active le mode révocation de la tâche en mettant celui-ci dans l'état prêt.

4.2. L'ordonnanceur et le contrôleur de charge

L'ordonnanceur natif de RT-Linux à priorité fixe a été remplacé par un ordonnanceur "Earliest Deadline".

Le contrôleur de charge est effectué par la fonction rt_contrôle_charge() dont le principe a été détaillé au paragraphe 3.3. Le pseudo-code est donné en annexe.

5. Conclusion

Notre travail s'intéresse aux applications temps réel comportant à la fois des tâches à contraintes strictes et des tâches à contraintes relatives et présente une méthode de prise en compte des surcharges permettant de minimiser l'occurrence de fautes temporelles ainsi que leurs conséquences sur le comportement de l'application : cette méthode s'appuie sur la notion de contrôleur de charge et sur un modèle de tâches multi-modes dédié à la prise en compte des conséquences des fautes temporelles et des suppressions d'exécutions.

Dans notre modèle, une tâche T est composée par un ensemble de quatre modes : le mode normal est activé au réveil de la tâche et correspond à l'exécution normale de la tâche. Il est notamment qualifié par un paramètre importance qui décrit le primordialité de son exécution au sein de l'application temps réel, un paramètre échéance qui décrit l'urgence de son exécution et un couple de propriétés de gestion d'exécution qui décrit comment son exécution peut être supprimée par le contrôle de charge. Les modes de survie Ajournement et Révocation sont des modes activés lorsque l'exécution de mode normal est respectivement ajournée (supprimée avant son début) ou révoquée (supprimée après son début). Enfin, le mode Faute Temporelle est activé lorsque l'un des trois modes précédents commet une faute temporelle.

A chaque réveil d'un nouveau mode normal, le contrôleur de charge évalue la laxité de chacun des modes normaux actifs afin de déterminer si le système tombe en situation de surcharge. Si oui, il utilise alors le paramètre importance pour résorber la surcharge en effectuant des suppressions d'exécution des modes normaux par ordre croissant d'importance. Pour chaque mode normal dont l'exécution est supprimée, il active le mode de survie Ajournement ou Révocation correspondant.

D'autres travaux traitent de contrôle de surcharge en utilisant un paramètre importance. [Jensen 85][Koren 95] présentent une méthode proche de la nôtre qui s'en diffère cependant par deux points :

- lorsqu'un surcharge est détectée, les suppressions d'exécutions de tâches s'effectuent de manière à maximiser une critère de performance ß, égal à la somme des exécutions réussies. Cette approche ne conduit pas, au moment d'une surcharge, à forcément favoriser, les exécutions courantes les plus importantes.
- le modèle de tâches utilisé ne permet pas de supporter les conséquences des suppressions effectuées.

Nous avons présenté l'implantation réalisée de notre modèle de tâches et du contrôleur d'ordonnancement au sein du système RT-Linux. Le choix de RT-Linux a été dicté notamment par la disponibilité des sources du système et également par la facilité de développement que permet la notion de module chargeable. Cette implantation nous permet dès maintenant d'évaluer les performances de notre méthode d'une part en mesurant le surcoût généré par le contrôle de charge, d'autre part en l'appliquant à une application de laminage composée d'un ensemble de 12 tâches périodiques liées par des contraintes de précédence [Kaiser 99]. On peut d'ores et déjà noter que l'association à chaque mode d'une tâche, d'une structure de processus lourd, impliquera sûrement un coût élevé en terme de commutation de tâches au moment des activations / désactivations de modes. La version prochaine de RT-Linux, intégrant la notion de processus léger, nous permettra de traduire notre modèle de tâches en allouant un processus léger par mode, ce qui sera de toute évidence plus performant.

Parallèlement, nous continuons d'affiner notre modèle de tâches de manière à prendre en compte le mode de survie Faute Temporelle et la variation dans le temps de la valeur des propriétés d'exécution des modes normaux. Ceci permettra de tenir compte des

suppressions déjà subies et d'interdire que les mêmes modes soient toujours victimes du contrôle de charge.

6. Bibliographie

[Barabonov 96] Barabanov et Yodaiken, "Real-Time Linux", Linux Journal, Mars. 96

[Delacroix 96] J. Delacroix, "Towards a satble Earliest Deadline scheduling algorithm", Real-Time Systems journal, Vol 10, number 3, may 1996, pp 236-291

[Delacroix 98] J. Delacroix, C. Kaiser, "Un modèle de tâches temps réel pour la résorption contrôlée des surcharges", RTS'98, 10-12 janvier 1998, Paris

[Epplin 97] Jerry Epplin, "Linux as an Embedded Operating System", Embedded Systems Programming, Octobre 97.

[Jensen 85] Jensen E.D., Locke C.D., Tokuda H., "A Time-Driven scheduling Model for Real-Time Operating Systems", Proceedings of 1985 IEEE Real-Time Systems Symposium, pp 112-122, 1985

[Kaiser 99] C. Kaiser, G. Stoffel, "Système d'acquisition et d'analyse en temps réel des signaux d'un laminoir RHENALU à NEUF-BRISACH", rapport Cedric 9901, 1999

[Koren 95]Koren G., Shasha D., "Dover : An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems", SIAM J. Comput., Vol 24, n°2, pp 318-339, 1995

[Liu 73] Liu C.L., Layland J.W., "Scheduling algorithms for multiprogramming in a hard real-time environment", Journal of the ACM, 20(1), pp 46-61, 1973

[Welsh 95] Matt Welsh, "Implementing Loadable Kernel Modules for Linux", Dr'Dobbs, mai 1995

7. Annexe : pseudo-code de la fonction rt_contrôle_charge()

```
void rt_controle_charge(void)
{
t = current_time();
surcharge = faux;

tant que toutes les tâches de la file prêtes classées par échéance croissante
n'ont pas été prises en compte
faire
     se placer sur la tete de la file des tâches pretes
     surcharge = faux;
     C_computation_time = 0;
     Pour toute tâche task de la file des tâches prêtes depuis ready_head
     faire
        C_computation_time = C_computation_time + task->rem_exec_time;
        Si (C_computation_time >= task->absolute_deadline - t;
        alors
           /* il y a surcharge; calculer sa valeur */
           Surcharge = vrai;
           val_surcharge := C_computation_time - task->absolute_deadline + t;
           task_fautive = task;
           break;
        fsi
     fait;

     Si (surcharge)
     alors
     /* résorber la surcharge si possible : on gere une file des taches pretes
   classees par importance croissante : c'est la file f_importance */
```

```
   Pour toute tâche task de la file des tâches prêtes telle que l'echeance est
   inferieure ou egale a celle de task_fautive
   faire
        chainer les tâches dans la file des tâches par importance croissante
        f_importance
   fait

     somme_suppression = 0;
     pour chaque tâche task de la file des tâches par importance croissante
     f_importance
     faire
      Si (task->properties[0] = ADJOURNABLE et task->state = RT_TASK_READY)
      alors
          /* la tache est ajournable (etat READY_AJ): ajourner la tâche */
          rt_task_adjourn (&task);
          somme_supression = somme_suppression + task->rem_exec_time;
      sinon
               Si (task->properties[1] = ABORTABLE et task->state =
                RT_TASK_PREEMPT)
                alors
                  /* la tache est revocable : revoquer la tâche */
                  rt_task_abort (&task);
                  somme_supression = somme_suppression + task->rem_exec_time;
                fsi
      fsi
      si somme_suppression >= val_surcharge
      alors
          surcharge = faux;
          break;
      fsi
     fait  /* liste f_importance */
     fsi /* si surcharge */
     si (surcharge = vrai)
     alors
         /* dans ce cas on est sorti parce que il n'y a plus de taches a
         supprimer mais la surcharge n'est pas resorbee */
         lever une alarme
     fsi

fait /* initial */

}
```