

LINUX

ET LA PROGRAMMATION TEMPS-RÉEL



RAPHAËL ARRIGONI, RARRIGONI@LINUXTRAINING.COM, [HTTP://WWW.LINUXTRAINING.COM](http://www.linuxtraining.com)
& FIORENZO.GAMBA@eif.ch, EIF-TÉLÉCOMMUNICATIONS



INTRODUCTION

Dans l'industrie informatique, il existe de nombreux systèmes d'exploitation temps-réel qui sont vendus par les entreprises à travers différents modes de distributions et de licences. Aujourd'hui, avec le monde de l'*open source* et plus particulièrement avec Linux et son implémentation d'interface de programmation de IEEE POSIX.1, il est possible d'écrire des applications temps-réel pour des systèmes embarqués sans avoir recours à un système d'exploitation propriétaire. Linux procure de formidables possibilités dans la programmation temps-réel mais il contient également certaines limites.

Nous allons donc explorer le fonctionnement de Linux face aux caractéristiques et aux exigences de la programmation temps-réel dont nous n'aborderons que la partie logicielle. Une application simple, une chaîne de fabrication de wagons-jouets en couleurs illustrera nos propos. Dans cette chaîne, les wagons doivent être aiguillés dans différentes directions en fonction de leur couleur.

Cet article est basé sur la version du noyau Linux 2.4-18.

LINUX ET LES CARACTÉRISTIQUES DU SYSTÈME TEMPS-RÉEL

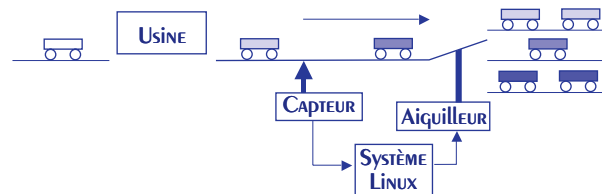
Un système est dit temps-réel s'il est capable de répondre à temps et tout le temps à un événement extérieur dans un intervalle temporel défini. Dans un système temps-réel, aucun dépassement du temps d'intervalle défini n'est acceptable car les conséquences d'un dysfonctionnement peuvent être dramatiques. Par exemple, la surchauffe d'un réacteur nucléaire peut entraîner de graves conséquences si il n'est pas détecté dans le temps défini. Cependant, un système temps-réel n'est pas obligatoirement un système rapide. Linux prouve qu'il peut être un système d'exploitation efficace et sûr dans ce domaine.

Le système temps-réel résulte de la combinaison de matériels, d'un système d'exploitation et d'applications. Enfin, un système temps-réel doit aussi être capable d'exécuter des tâches dans des temps précis, et ceci de manière périodique ou non-périodique.

EXEMPLE: Aiguillage de wagons

Dans un centre de triage, les wagons qui sortent d'une usine de peinture sont automatiquement aiguillés dans différentes directions en fonction de leur couleur.

La contrainte de temps liée au système temps-réel suppose que le wagon sortant de l'usine de peinture soit dans un premier temps détecté par le capteur. Le système doit dans un second temps lire sa couleur et prendre une décision sur l'aiguillage avant que le wagon n'arrive dessus. Dans cet exemple, le système doit toujours être plus rapide que la durée de trajet du wagon du capteur à l'aiguillage.

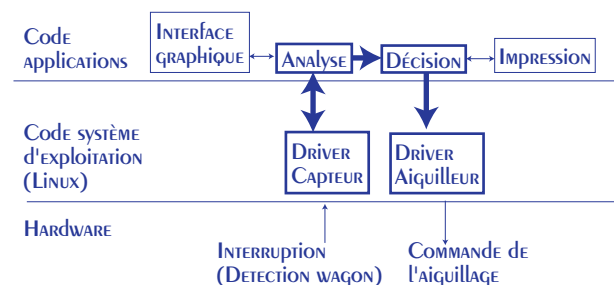


Si le seul travail du système est de gérer la sortie des wagons, il est plus facile de garantir un temps de réponse. Cependant, en réalité, de tels systèmes gèrent plusieurs événements extérieurs et de multiples applications sont en exécution à des niveaux de priorités différents.

DÉTECTION ET SÉQUENCE D'EXÉCUTION

Dans un système temps réel, il est nécessaire d'identifier tout le code qui va être exécuté entre la détection de l'événement extérieur et l'action.

Dans notre exemple, la détection des wagons ainsi que celle de la couleur peuvent se faire à l'aide d'un capteur qui génère une interruption pour chaque wagon qui passe. L'information est ensuite dirigée dans l'application qui prend la décision nécessaire pour l'aiguillage du wagon.



D'une part, la gestion des interruptions et l'accès matériel dans un système se font au moyen du code dans l'espace noyau. D'autre part, la gestion des événements se fait dans l'espace utilisateur qui peut être une application.

Dans le design d'un système temps-réel, il est important que le programmeur identifie les chemins critiques qui sont des séquences de code qui ont besoin d'être exécutées afin de répondre à un événement extérieur. Les chemins critiques peuvent se croiser et ont souvent des niveaux de priorités différents.

TEMPS DE RÉPONSE

Lors de la détection d'une interruption, le code se trouvant sur le chemin critique doit pouvoir être exécuté à sa plus haute priorité.

Avec Linux et sa librairie *pthread*, il est possible de pouvoir utiliser le processeur à plein régime et d'arrêter toute autre tâche à l'exception des routines d'interruption afin de répondre à un événement extérieur.

Le temps de réponse se situe donc dans l'exécution de toutes les parties de code ainsi que dans d'autres activités qui peuvent avoir un impact sur la situation. Il peut s'agir de:

- code critique partagé par des *threads* de différentes priorités;
- exécution d'autres routines d'interruption;
- appels système par d'autres *threads*.

Ces derniers points peuvent avoir des conséquences graves dans un système temps-réel s'ils ne sont pas pris en considération et nous allons voir comment Linux réagit dans ces domaines.

CODE CRITIQUE PARTAGÉ

Précisons que Linux est un système d'exploitation multi *threads*. Un *thread* est l'entité la plus petite qui est exécutée sur le processeur par l'ordonneur. Chaque *thread* pourra être exécuté par le noyau en fonction d'un algorithme qui prend en compte la priorité du *thread* et son intervalle de temps d'exécution.

Chaque *thread* peut exécuter des fonctions. Le problème survient lorsqu'une fonction peut être exécutée simultanément par 2 *threads* de façon interlacée. La fonction peut alors avoir des comportements différents. Pour éviter des conséquences fâcheuses, les fonctions doivent être sérialisées en utilisant des verrous du standard POSIX.1.

Dans un système temps-réel, il est important d'utiliser des verrous avec l'héritage de priorité ou à plafond afin d'éviter des situations d'inversion de priorité.

Une situation d'inversion de priorité survient lorsque:

- un *thread* à basse priorité commence à exécuter du code critique;
- un deuxième *thread* à haute priorité veut exécuter la même fonction mais il reste bloqué à l'entrée du code critique parce que le verrou est verrouillé;
- un troisième *thread* de moyenne priorité exécute une fonction pour un temps indéterminé.

Dans cette situation le *thread* de haute priorité est bloqué sur le verrou parce que le *thread* de moyenne priorité est en exécution et empêche le *thread* de basse priorité de finir l'exécution du code critique.

Dans l'implémentation de POSIX.1 sur Linux, ces mécanismes de priorité d'inversion ou à plafond ne sont à l'heure actuelle pas implémentés. Linux est donc limité dans ce domaine.

Dans le design de votre système, il est impératif d'identifier le code qui risque d'être partagé par des *threads* à différentes priorités et si possible de le dupliquer.

EXÉCUTION D'AUTRES ROUTINES D'INTERRUPTION

Le travail principal de l'ordonneur consiste à choisir en fonction d'un algorithme le *thread* qui peut être exécuté sur le processeur. Les entités qui peuvent être exécutées sans l'accord de l'ordonneur sont uniquement celles relatives aux routines d'interruptions qui vont avoir un impact direct sur le temps de réponse du système.

En général, les routines d'interruptions sur Linux sont très courtes. Le code se trouve dans une autre entité qui peut être une *tasklet* ou un *bottom-half* gérée en fonction d'une priorité par l'ordonneur.

Les appels systèmes sont exécutés dans le contexte du *thread* qui fait l'appel.

Dans la personnalisation de Linux et de ses modules, il est impératif de connaître d'une part les détails des différents modules installés, et d'autre part l'implémentation des routines d'interruption, notamment leur temps de réponse et de cadences maximum d'exécution. Cela évite des surprises sur les temps de réponse du système temps-réel.

APPELS SYSTÈMES

Par défaut, le noyau de Linux 2.4 n'est pas préemptif. Un *thread* qui fait un appel système non-bloquant devra attendre le retour de l'appel afin de permettre l'exécution du *thread* suivant. Cette contrainte peut rajouter des temps indéfinis au temps de réponse.

Aujourd'hui, le fait que le Linux ne soit pas préemptif dans l'espace noyau constitue une lacune fâcheuse lorsqu'il est utilisé dans un système temps-réel. Cependant, des travaux sont en cours pour le rendre préemptif et dans la version du noyau 2.5, tous les appels systèmes devraient être réentrants.

CONCLUSION

A l'heure actuelle, Linux est très utilisé dans le monde des systèmes embarqués. Il reste néanmoins discret dans celui du temps-réel car il n'a pas encore gagné la confiance des industriels. Ces derniers préfèrent utiliser des variantes de Linux ou encore des systèmes propriétaires exempts des lacunes qui handicapent Linux dans ce domaine.

Le système d'exploitation Linux demeure donc un petit concurrent dans la programmation temps-réel mais attendons nous à ce que cette situation change très vite;-).■