Rapport de stage Ingénieur

-

Implémentation d'un ordonnanceur temps réel sur plateforme multi-cœur hétérogène

-

BELPOIS Vincent

2023

Table des matières

Présentation du stage

0.1 Le L.I.A.S.

0.2 Le sujet du stage

Mon stage s'intéresse à l'implémentation d'un ordonanceur sur plateforme hétérogène[1]

1 Systèmes d'exploitation compatibles avec la carte ROCK960

1.1 Présentation de la carte de développement

1.2 Installation d'un système d'exploitation

1.2.1 Installation d'une image précompilée

Image Linux ubuntu fournie par 96boards

1.2.2 Compilation de Linux depuis le code source

Ou est le code source linux ?
Comment changer de version du code cloné.

1.2.3 Compilation croisée

Toolchain : qu'est ce que c'est
Variables d'environement
Réalisation de scripts linux pour accélérer le développement
Copy de l'image du noyau pour faire encore plus rapide

1.3 Etude des versions de Linux compatibles

1.4 Comment Linux gère le suport d'un processeur

1.5 Etude

2 LITMUS^{RT}

2.1 Présentation de LITMUS^{RT}

2.2 Présentation de *feather-trace*

2.3 Implémentation d'un ordonanceur EDF partitioné

2.3.1 Algorithme considéré

On cherche alors pour commencer à implémenter un algorithme d'ordonnancement simple afin de se familiariser avec les méthodes et fonctions fourni par LITMUS^{RT}. J'ai donc choisi un algorithme partitioné pour la simplicité d'ordonnancement par processeur que cela offre. Un algorithme EDF (*Earliest Deadline First*) est alors choisi pour la simplicité du choix de la tache a exécuter. Comme son nom l'indique, on choisi à chaque instant la tache ayant l'échéance la plus proche. On nommera par la suite cet algorithme P-EDF (*Partitionned Earliest Deadline First*).

Pour montrer le fonctionnement de cet algorithme, si l'on se place sur un même processeur, on peut visualiser l'éxécution de deux tache periodiques :

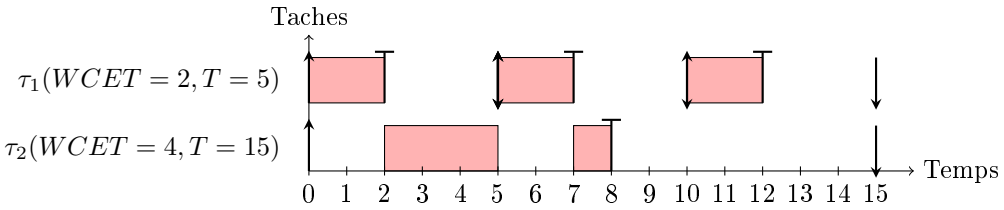

FIGURE 1 – Exemple de EDF à 2 taches

On a ici une première tache τ_1 avec un pire temps d'éxécution (*Worst Case Execution Time*) de 2 et une période de 5, et une seconde tache τ_2 avec un pire temps d'éxécution de 4 et une période de 15. On a alors préemption de la τ_2 à $t = 5$ afin d'éxécuter τ_1. Cela est dû au réveil de la tâche τ_1 (représenté par la flêche montante) et à la date d'échéance plus proche de cette dernière.

2.3.2 Implémentation

La construction d'un plugin d'ordonnancement nécessite la déclaration d'un module au sens de Linux. Pour Linux un module est un élément de code qui peut être chargé dynamiquement lors de l'exécution du système d'exploitation. Un module permet alors d'étendre les fonctionnalités du noyau, il a donc ont accès aux fonctions du noyau, à ses resources et peut aussi réaliser des appels systèmes.

Pour que notre nouvel ordonnanceur soit reconnu par le noyau Linux modifié (LITMUS^{RT}) ; il faut alors déclarer une fonction d'initialisation :

```
1 #include <linux/module.h> // used for calling module_init()
2
3 static int __init init_p_edf(void)
4 {
5     return 0; // indicates a successful initialisation
6 }
7
8 module_init(init_p_edf); // specify the entry point of the module
```

On peut alors enregistrer ce fichier sous le nom sched_p_edf.c pour suivre la nomenclature des autres ordonnanceurs fournis avec avec LITMUS^{RT}. Ce fichier est enregistré dans le dossier llinux/litmus. On peut alors modifier le fichier Makefile de ce dossier afin de l'ajouter au fichier à compiler :

```
1 #
2 # Makefile for LITMUS^RT
3 #
4
5 obj-y = sched_plugin.o litmus.o \
6         preempt.o \
```

```
        litmus_proc.o \
        budget.o \
        clustered.o \
        jobs.o \
        sync.o \
        rt_domain.o \
        edf_common.o \
        fp_common.o \
        fdso.o \
        locking.o \
        srp.o \
        bheap.o \
        binheap.o \
        ctrldev.o \
        uncachedev.o \
        sched_gsn_edf.o \
        sched_psn_edf.o \
        sched_pfp.o \
        sched_p_edf.o

obj-$(CONFIG_PLUGIN_CEDF) += sched_cedf.o
obj-$(CONFIG_PLUGIN_PFAIR) += sched_pfair.o

obj-$(CONFIG_FEATHER_TRACE) += ft_event.o ftdev.o
obj-$(CONFIG_SCHED_TASK_TRACE) += sched_task_trace.o
obj-$(CONFIG_SCHED_DEBUG_TRACE) += sched_trace.o
obj-$(CONFIG_SCHED_OVERHEAD_TRACE) += trace.o

obj-y += sched_pres.o

obj-y += reservations/
```

On place notre fichier à compiler sous mot clé `obj-y` pour signifier que l'on veut ce module compilé et inclus lors de la compilation du noyau Linux.

- Montrer ce qui est propre a la définition du module (sauf si je l'explique dans la partie sur le noyau Linux)
- Montrer l'emplacement des fichiers que l'on va créer dans le noyau (avec une hiérarchie des fichiers, un arbre)
- Montrer les modification du make file

Références

[1] Antoine Bertout, Joël Goossens, Emmanuel Grolleau, and Xavier Poczekajlo. Workload assignment for global real-time scheduling on unrelated multicore platforms. In *Proceedings of the 28th International Conference on Real-Time Networks and Systems*, pages 139–148, 2020.

Table des figures

Glossaire

plateforme hétérogène Système formé d'un ensemble de processeurs différents. 3

processeur Ca c'est la définition. 5