Temps réel mou et Linux
M1 I&ISC 2016-2017

P. Andry

Prélude

Cette étude se place dans le cadre de la réalisation d'un ensemble applicatif se déroulant sur un système Linux classique. Etant donné le système d'exploitation (noyau linux classique 3.xx), nous ne pouvons pas garantir de temps réel dur. Cependant, par des techniques de programation simples en utilisant l'interface POSIX, nous pouvons garantir une mesure du temps et des certifications temps réel mou utiles pour le déroulement d'applications informatiques.

Cas pratique, temps réel mou et linux embarqué

L'architecture cible supporte le multithreading, mais ne supporte pas le temps réel dur (par exemple Linux Angstrom). En remettant en avant une partie des contraintes propres à chaque tâche, les concepteurs (deux anciens brillants étudiant du Cergy-Pontoise Institue of Technology) qui développent en C POSIX ont dégagé un schéma d'architecture suivant :

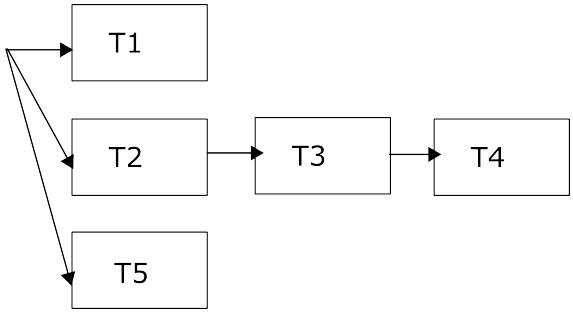

Figure 1: Chaque tâche, notée de T1 à T5 est encapsulée par un thread POSIX. Les threads s'executent à la fois en concurrence (quand ils sont sur des lignes différentes) et à la fois de manière sérielle (quand ils sont sur une même ligne) dans l'ordre indiqué par les flêches. Ici, T4 doit s'executer une fois T3 achevé, qui lui même doit attendre que T2 soit achevé. En parallèle, T1 et T5 s'executent sans contraintes. Plusieurs executions de T1 ou de T3 peuvent avoir lieu pendant que la ligne T2→T3→T4 s'execute.

Dans le projet actuel, nous ne nous concentrerons pas sur les algorithmes spécifiques de chaque tâches (ce peut être du traitement d'image, du contrôle robotique, des communications réseau, des procédures d'interface, etc.), mais sur les routines globales et une partie du code de chaque tâche qui permet de contrôler l'ordonnancement et les échéance au sein de l'application.

rappelez vous..

- Rappelez comment programmer un thread pour que la tâche qu'il encapsule se répète indéfiniment (comportement souhaité pour chaque tâche).
- Rappelez le comportement d'un thread quand il est *attaché* ou *détaché*.

- Comment mettre en attente le thread principal, jusqu'a ce que tous ses fils soient terminés ?
- Comment indiquer à chaque thread comment se terminer quand l'appelant se termine. ?
- A quoi servent Mutex et Sémaphores ?
- Rappelez-vous ce qu'est un mécanisme de *rendez-vous* en POSIX.

A vous :

- Proposez une méthode simple utilisant les outils POSIX pour sérialiser des threads (par exemple T2,T3,T4).Discutez les avantages et inconvénients de votre méthode. Implémentez.
- Etendez votre solution pour qu'elle puisse *sérialiser* plusieurs lignes *en parallèle*,. Les numéros (ou les noms) des threads seront contenu dans un ficher que vous utiliserez pour créer et ordonnancer les threads. Par exemple :

```
NB_LINES = 3
LINE 1 : T1 -> End
LINE 2 : T2 -> T3 -> T4 -> End
LINE 3 : T5 -> END
```

- Proposez une méthode POSIX pour signaler si des échances sont manquées, en fonction de la contrainte temporelle de chaque ligne choisie par l'utilisateur. Par exemple, on souhaitera que T2 puis T3, puis T4 soient exécutées en moins de 300ms. Si l'échéance d'une ligne est manquée, les taches de la ligne en cours seront stoppées et la ligne relancée du début.
- Mettez a jour votre fichier pour qu'il comporte les échéances voulues pour chaque ligne:

```
NB_LINES = 3
LINE 1 : T1 -> End 300
LINE 2 : T2 -> T3 -> T4 -> End 1000
LINE 3 : T5 -> END 100
```

A rendre : note sur 5 points

Vous rendrez un mini-rapport de 1 page qui explique les solutions utilisées pour réaliser le séquencement des lignes ainsi que la mesure du temps et la signalisation des échéances manquées. Vous joindrez le code source en annexe du mini-rapport.

Approfondissement (5 points bonus):

- Proposez une version qui signale quelle tâche se trouve en exécution lorsque l'échéance de la ligne est ratée. (+2pts)
- Proposez une version qui permet de charger le code des tâches "à chaud", en utilisant les fonctions de la librairie `dlfcn.h`, `dlopen, dlsym, dlclose,` *etc*. Les tâches T1, T2, etc.. sont ainsi des sections code objet compilé séparément, à la manière de pluggins dont le code sera chargé "a chaud" lorsque le programme en aura besoin (le déchargement du plugin interviendra a la fin du programme, ou a la demande de l'utilisateur). (+3 pts)