TD - Programmation Temps Réel

L'objectif des séances de TD est de se familiariser au développement et à l'analyse d'une application temps réel à partir du système d'exploitation temps réel *Trampoline*.

Le développement se fera directement dans le Terminal de Linux. Il est conseillé de travailler directement sur votre ordinateur personnel.

Vous trouverez sur le *campus* :
— Le document *guide_Trampoline.pdf* qui comporte, entre autres, les instructions pour installer Trampoline et développer une application.
— Une archive *appli_base* dans laquelle se trouve les sources d'une application de base.

Dans ce document, les questions et manipulations à réaliser sont indiquées par le symbole →.

1 Contrôle d'un feu tricolore

On cherche ici à développer une application permettant de contrôler les feux d'un feu tricolore. Le fonctionnement du feu est le suivant :

- Le feu doit être au rouge pendant 5 secondes puis passer au vert.
- Le feu doit être au vert pendant 4 secondes puis passer à l'orange.
- Le feu doit être à l'orange pendant 1 seconde puis passer au rouge.

Chacun des feux du feu tricolore est associé à une LED de la carte d'évaluation.

En plus de la gestion des feux, l'application doit réaliser un calcul pendant 0.5 seconde. *Le calcul sera indiquer par l'intermédiaire de la LED bleu.*

1.1 Version 1 : Application de base

Pour la première version, l'application doit respecter l'architecture fonctionnelle de la Figure 1.

- La tâche `task_manage` doit gérer le passage d'un feu à l'autre puis réaliser le calcul.
- Les tâches `task_red`, `task_orange` et `task_green` doivent respectivement modifier l'état de la LED rouge, orange et bleu.

→ Déterminer la priorité des tâches et la période de l'alarme.

→ Coder l'application.

1.2 Version 2 : Ajout du mode maintenance

On veut maintenant ajouter un mode maintenance dans lequel le feu orange clignote (il s'allume pendant une seconde et s'éteint pendant 1 seconde). Le passage d'un mode à l'autre se fait via une ISR qui se déclenche par l'appui sur le bouton poussoir.

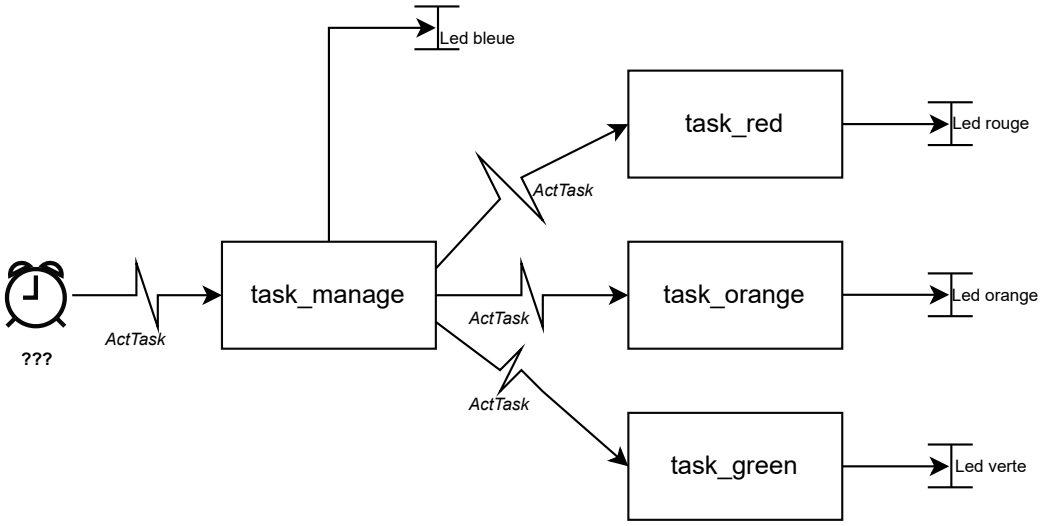

FIGURE 1 – Architecture fonctionnelle 1 du feu tricolore

La tâche `task_manage` peut maintenant éteindre les feux et ne réalise pas le calcul lors du mode maintenance.

→ Donner l'architecture de l'application logicielle.

→ Coder l'application.

→ Donner le diagramme de Gantt de l'application pendant le mode maintenance. Vérifier l'exécution de l'application avec GDB.

→ Quelle remarque peut-on faire sur les changements de contexte de l'application ?

→ Lire les *slides* 15 à 16 du document *services_Trampoline.pdf*.

→ Modifier l'application afin d'éviter les changements de contexte inutiles.

2 Les tâches étendues

→ Lire les *slides* 23 à 35 du document *services_Trampoline.pdf*.

On se place dans l'architecture logicielle de la Figure 2.

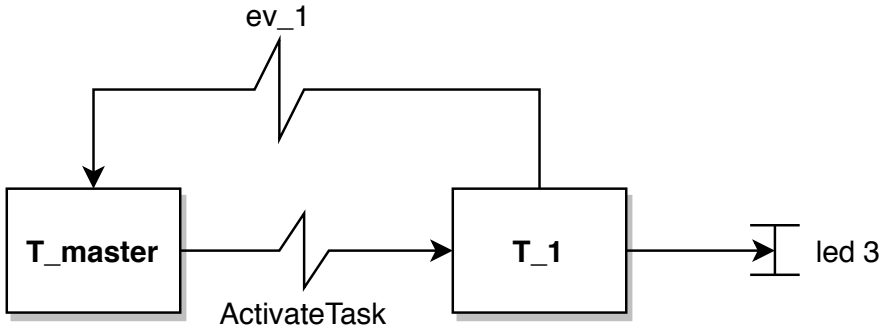

FIGURE 2 – Architecture logicielle de l'application 3

Au démarrage de l'application, `T_master` active la tâche `T_1` puis va indéfiniment faire :
— Attendre un événement.
— Activer la tâche `T_1`.

Une fois activer, la tâche `T_1` doit :

- Allumer la led 3
- Attendre 500 ms
- Éteindre la led 3
- Attendre 500 ms
- Envoyer un événement à `T_master`.
- Se terminer.

On impose que la priorité de `T_master` est supérieure à celle de `T_1`.

→ Donner le diagramme de Gannt de l'application.

→ Coder l'application et exécuter là. Vérifier que l'exécution est bien celle attendue.

Maintenant la tâche `T_master` doit au démarrage activer deux tâches supplémentaires : `T_2` et `T_3`. Elle doit ensuite indéfiniment activer la tâche qui vient de lui envoyer un événement.

Le code de `T_2` et `T_3` est identique à celui de `T_1` sauf que chacune des tâches doit gérer une LED différente et envoyer un événement différent.

→ Coder l'application et exécuter là.

3 Les alarmes

→ Lire les *slides* 36 à 50 du document *services_Trampoline.pdf*.

3.1 Un chenillard

On veut réaliser un chenillard comme présenté sur le chronogramme de la Figure 4. On impose que l'application soit composée d'une tâche `T_chase` et de quatre alarmes.

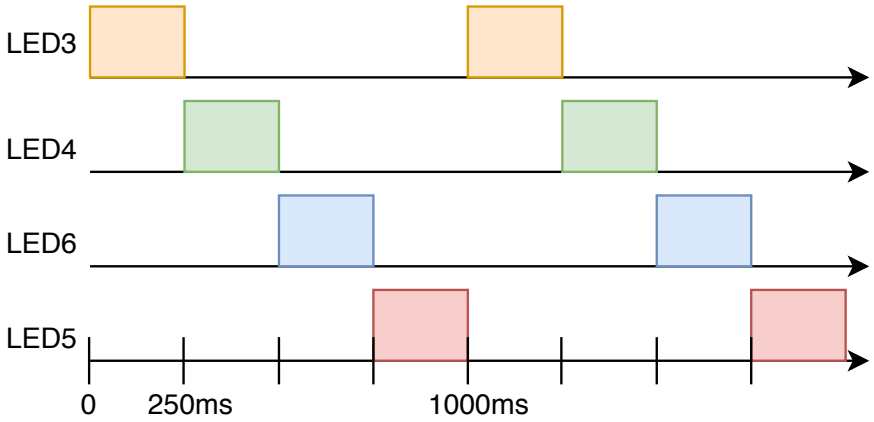

FIGURE 3 – Chronogramme du chenillard

→ Dessiner l'architecture logicielle de l'application.

→ Programmer et exécuter l'application.

→ Modifier votre application afin que l'appui sur le bouton poussoir inverse le sens du chenillard.

4 Les ressources

La mise en place de ressource permet de protéger des sections critiques. On peut par exemple penser à l'accès concurrent à une variable en écriture qui peut engendrer une corruption de donnée.

→ Lire les *slides* 57 à 82 du document *services_Trampoline.pdf*.

4.1 Une application

Pour mettre en évidence l'utilisation de ce type de mécanisme, on va utiliser l'application présentée sur la Figure 5.

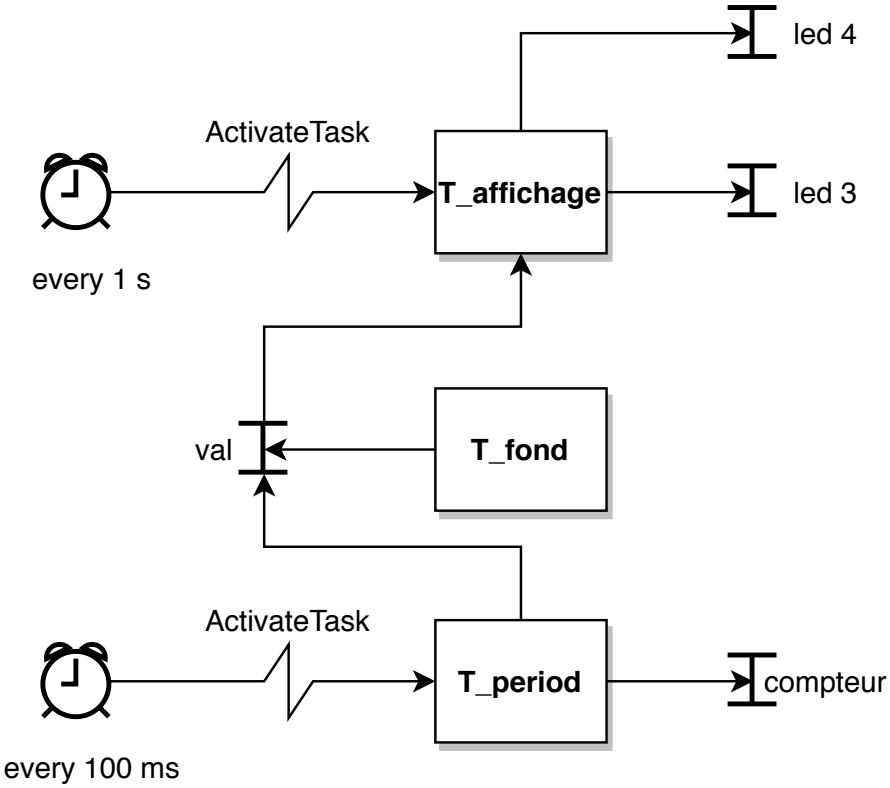

FIGURE 4 – Architecture logicielle de l'application

L'application est composée de deux variables `volatiles` globales et de trois tâches :
— `T_affichage` a une priorité de 10 et est activée toutes les secondes. Si la valeur de la variable `val` se trouve dans l'intervalle attendu alors la LED 3 doit être allumée, sinon c'est la LED 4 qui doit l'être.
— `T_fond` a une priorité de 1 et est activée en permanence. Elle doit incrémenter puis décrémenter la variable `val` indéfiniment.
— `T_period` a une priorité de 5 et est activée toutes les 100 ms. Elle doit incrémenter la variable `compteur`, puis si `compteur` est paire elle doit incrémenter `val` sinon elle doit la décrémenter.

→ Comment doit évoluer la variable `val` ?

→ Programmer et exécuter l'application. Que peut-on remarquer ?

4.2 Utilisation d'une ressource

On propose de mettre en place une ressource pour protéger la variable `val`.

→ Compléter le document de synthèse des services de Trampoline pour les appels systèmes liés à la gestion des ressources.

→ Compléter le document de synthèse des objets OIL pour l'objet *RESOURCE*.

→ Modifier et tester votre programme.

→ En analysant le fichier *tpl_app_config.c*, quelle est la valeur de la priorité des différentes tâches et de la ressource ? Cela correspond t-il au protocole IPCP ?

4.3 Utilisation d'une ressource interne

Une ressource interne est automatiquement prise quand la tâche qui y est associée à le CPU.

→ Modifier le code pour que `T_fond` prenne la ressource interne.

→ Programmer et exécuter l'application. Que peut-on remarquer ?

→ Modifier le code pour que ne soit plus dans une boucle infinie mais s'appelle elle-même via l'appel système `ChainTask()`.

→ Expliquer ce qu'il se passe.