Projet IMA 4

Projet P28 :

Optimisation des TP de temps réel

Etudiants :
Gasnier Damien
Bonvalet Quentin

Encadrant :
Blaise Conrard

Date :
31/01/2013 au 07/05/2013

Sommaire

Introduction

Contexte

Pour l'année 2012/2013, les TP de temps-réel au 1er semestre de quatrième année d'IMA à Polytech Lille prennent au total 12 heures (trois fois quatre heures). Nous utilisons Linux RTAI et pouvons expérimenter les notions de programmes temps-réels avec des programmes écrits en langage C.

Les erreurs syntaxiques sont détectées à la compilation et peuvent ainsi être corrigée relativement rapidement. Mais les erreurs d'écritures ou oublis dans le code entraînent généralement, en temps-réel, des « plantage » de l'ordinateur sans aucun message d'erreur.

Afin d'éviter de devoir redémarrer l'ordinateur lors de ce genre d'erreur, il nous a été proposé de concevoir une solution. Celle-ci devrait également permettre de cibler la cause de l'erreur plus facilement, particulièrement les erreurs « classiques », ce qui permettrait aux élèves de gagner en temps et en efficacité, ainsi que d'alléger la frustration d'avoir à redémarrer l'ordinateur pour une simple erreur pouvant être corrigée en quelques secondes.

Cette outil devrait également permettre d'améliorer la compréhension du module temps-réel en offrant la possibilité de visualiser la séquence d'ordonnancement des tâches à l'aide d'un chronogramme.

Solution envisagée

Un programme devrait alors mémoriser toutes les informations relatives à chaque tâche (période, date de démarrage, priorité, ect …) pour pouvoir simuler leurs exécutions ainsi que la modification des entrées / sorties qu'elles entraînent.

Le programme devrait aussi retourner un chronogramme de l'exécution des tâches et de l'évolution des signaux d'entrées / sorties.

Il a été convenu avec notre encadrant que l'analyse de code pur n'était pas forcément ni la solution la plus simple, ni la solution la plus intéressante au niveau de notre compréhension du temps-réel.

Il a donc été convenu de créer un programme de simulation qui inclurait le programme de l'étudiant, sans les librairies RTAI. Les fonctions RTAI normalement appelées par le programme de l'étudiant seraient alors écrites dans le programme de simulation et permettrait d'exécuter le programme étudiant non pas avec le noyau temps-réel RTAI, mais simplement par l'intermédiaire d'un programme C sous linux.

Définition du cahier des charges

Contraintes

Le programme de simulation doit s'adapter au code de l'étudiant et pas l'inverse. L'étudiant ne doit rien avoir de spécial à rajouter dans son code et peut écrire son programme sans aucunes contraintes de plus que celles imposées par le langage C et les librairies qu'il souhaite utiliser.
La notion de dynamisme du programme de simulation est donc importante puisque l'étudiant peut définir autant de tâches qu'il le souhaite et leur donner un nom désiré.
Le programme de simulation devrait aussi pouvoir être utilisé dans le cas où les TP de temps-réel viendraient à être changé et utiliser d'autres fonctions RTAI qui n'avaient jusqu'ici pas été utilisées.

Le programme de simulation doit être exploitable d'ici la fin du semestre, c'est à dire vers le 7 Mai 2013. Il nous est impartit 40 heures de projet durant lesquelles il nous est également demandé de tenir à jour un wiki [1], de produire un rapport et une vidéo de présentation. Une soutenance de 10 minutes pour ce projet à lieux le 7 Mai 2013.

Libertés

Il nous est laissé le choix du ou des langages pour construire le programme de simulation et la production du chronogramme.
Nous sommes également libres dans notre organisation pour réussir à mener à bien ce projet en répondant correctement aux attentes.

Réalisation du projet

Importation du code étudiant

Nous avons créé un fichier « Makefile », un script Unix shell (Bash) à lancer pour compiler le programme de simulation.
Ce script permet de créer un fichier « FichierTP.h », qui est en fait le code que l'étudiant veut tester, auquel quelques modifications sont apportés grâce à la commande suivante :

```
cat prog_etudiant.c | sed -e "s/RTIME/int/g" |grep "#include" -v |grep "MODULE_LICENSE" -v | sed -e 's/^\(.*rt_task_init\)\(.*,\)\(.*,\)\(.*,\)\(.*,\)\(.*,\)\(.*\)/\1\2\&\3\4\5\6\7\8/' > FichierTP.h
```

Explication de la commande :

`cat prog_etudiant.c > FichierTP.h` : Le programme C de l'étudiant est copié et mis dans un FichierTP.h qui est alors créé.

`| sed -e "s/RTIME/int/g"` : Les variables de type RTIME, type de variable de la librairie RTAI, sont remplacées par de simples entiers.

`|grep "#include" -v` : On ne prend pas les librairies déclarées par le programme étudiant.

`|grep "MODULE_LICENSE" -v` : On ne prend pas la ligne d'inclusion de licence GPL, qui est déclarée lorsqu'on utilise les modules dérivés du noyau Linux. Cette ligne est présente dans les programmes vu en TP et provoque une erreur de compilation puisque nous n'utilisons pas les librairies temps-réel.

`| sed -e 's/^\(.*rt_task_init\)\(.*,\)\(.*,\)\(.*,\)\(.*,\)\(.*,\)\(.*\)/\1\2\&\3\4\5\6\7\8/'` : Permet de rajouter un « & » devant le nom des taches passées en argument de la fonction d'initialisation « rt_task_init ». Cela permet de passer en argument non plus une tâche, mais un pointeur sur cette tâche.

Mémorisation des tâches

Nous avons créé une structure pour pouvoir mémoriser dans un premier temps, au moins le numéro de la tache, sa première date d'exécution, sa priorité, et sa période puisque nous avons commencé par étudier le problème dans le cadre d'une exécution périodique comme nous avions vu en TP.

Nous avons repris le nom « RT_TASK » pour notre structure puisqu'avec RTAI chaque tâche demande d'être déclarée ainsi : « static RT_TASK idtache ; »

Toutes ces informations relatives aux tâches sont normalement déclarées grâce à deux fonctions RTAI :

int rt_task_init(RT_TASK * t, void (*callback)(), int param, int taille_pile, int priority, int drapeau , int signal)

int rt_task_make_periodic(RT_TASK * t, int now, int tick_period)

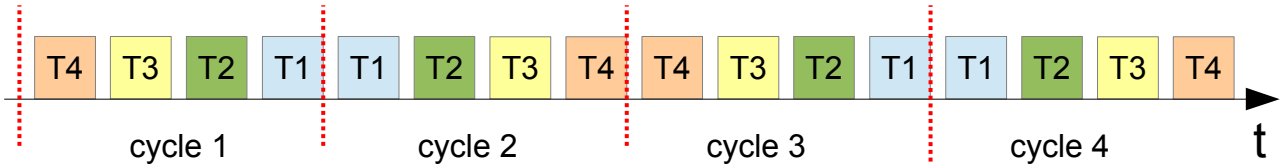
On peut alors mémoriser un pointeur sur chaque tâche créée (RT_TASK * t), avec la priorité (priority), la première date de réveil (now) et la période (tick_period)

L'argument « (*callback)() » est aussi utilisé pour mémoriser le nom de la tâche, qui est en fait une fonction du programme de l'étudiant, à laquelle il a donné le nom qu'il souhaitait. Cette technique permet de pouvoir appeler les tâches afin de les exécuter.
« (*callback)() » est ainsi stocké dans « nom_fct » dans la structure « RT_TASK », et grâce au pointeur « t » sur cette structure, on peut appeler la fonction, la tâche, ainsi :
« t -> nom_fct( ) ; »

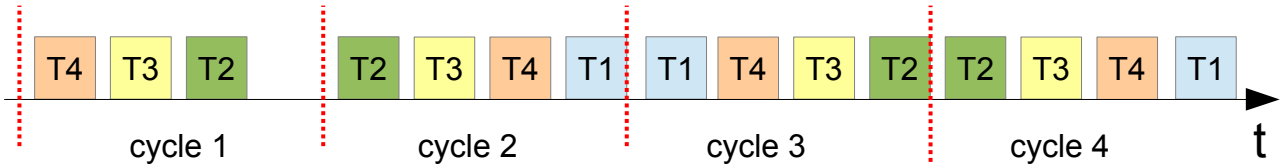
Ayant de ce fait toutes les informations nécessaires sur chaque tâches, il faut maintenant les appeler dans l'ordre où elles le sont en temps-réel.

Tests sur un ordonnanceur RTAI

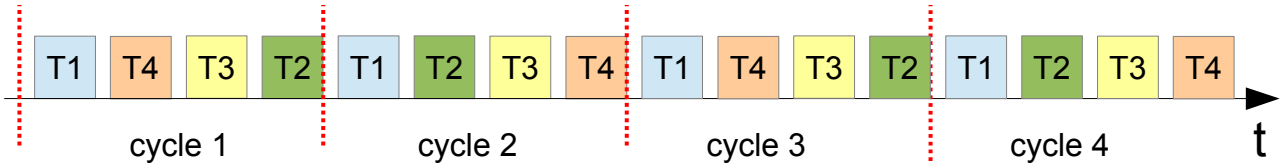
Pour vérifier comment les tâches sont effectivement appelées avec l'ordonnanceur de RTAI, nous avons réalisé plusieurs tests. Par exemple :



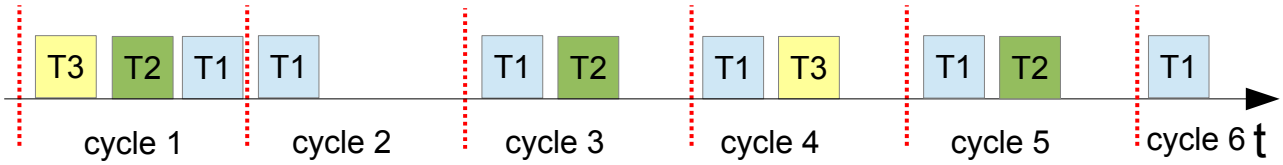
Ce premier test a été réalisé en mettant toutes les priorités, les périodes et les dates de réveils égales. On peut voir qu'on commence par exécuter la tâche la définie en dernier dans le programme. La tâche exécutée ensuite est la tâche déclarée avant. Puis dans le cycle suivant cet ordre est inversé.



Ce deuxième test a été réalisé en mettant les même paramètres à toutes les tâches, sauf T1 qui possède une première date de réveil égale à sa période. On voit que T1 n'est pas exécutée au premier cycle. Elle est exécutée en dernier au deuxième cycle. Ensuite on retrouve le même comportement qu'au premier test.



Ce troisième test a été réalisé en mettant les même paramètres à toutes les tâches, sauf la tâche T1 qui a maintenant une priorité plus basse que les autres. On peut voir que T1 est alors exécutée au début de chaque cycle.



Ce quatrième test a été réalisé en mettant les même paramètres à toutes les tâches, sauf la période. La tâche T2 possède une période deux fois plus grande que T1 et la tâche T3 possède une période trois fois plus grande que T1. La tâche T1 est exécutée à tout les cycle, T2 est exécutée un cycle sur deux et T3 est exécutée un cycle sur trois. Il est aussi intéressant de remarqué que T1, qui est définie en premier, est exécuté en dernier au premier cycle, comme dans le premier test. Mais T1 est ensuite toujours exécutée en premier.

Nous pouvons donc en conclure qu'une tâche s'exécute à partir de sa date de réveil, puis à chaque prochaine date de réveil, qui est la période de la tâche additionnée à la date de sa dernière exécution. Dans le cas où plusieurs tâches doivent être exécutées à la même date, c'est la tâche la plus prioritaire qui est exécutée. Dans le cas de priorités égales, c'est la tâche la dernière exécutée qui est alors exécutée de nouveau.

Ordonnancement

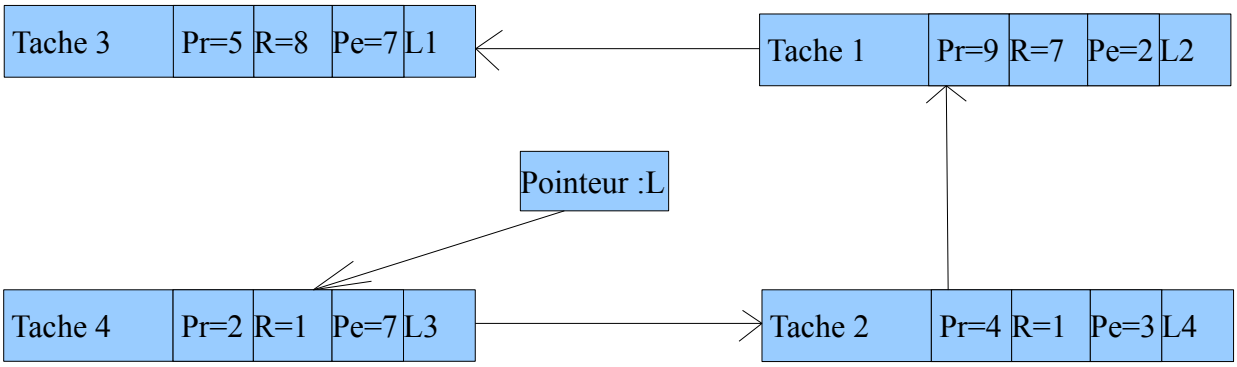
Réflexion méthode ordonnancement:

Nous avions d'abord pensé à ceci avant de faire les tests : trouver la tâche avec la prochaine date d'exécution la plus proche. Puis dans le cas de dates identiques, trouver la plus prioritaire. Puis la plus âgée.
Malheureusement, ceci est non compatible avec un ordonnancement tourniquet inversé mis en évidence lors des tests.

Autre méthode dont l'objectif: déterminer quelle tâche exécuter .
Solution : Liste chaînée triée (simple) + déplacement pointeur pour garder le 1er élément comme tâche à exécuter. Cette méthode permet de reproduire un ordonnancement tourniquet inversé.

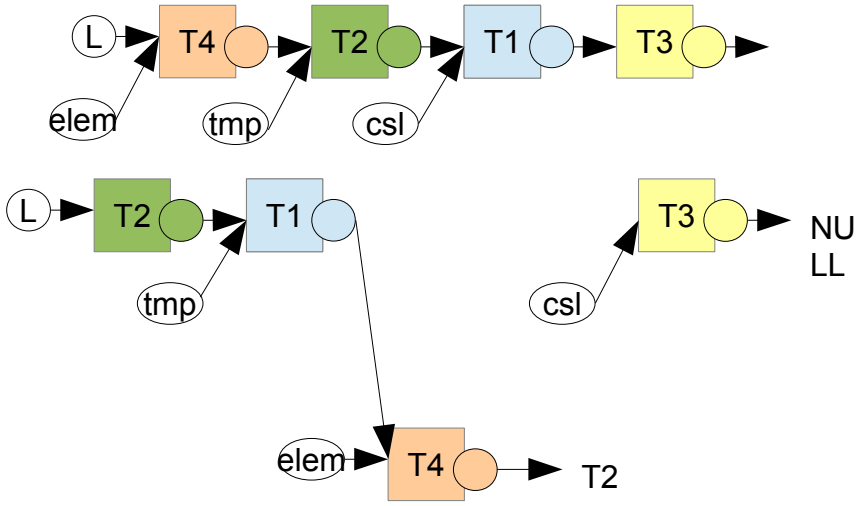
Création d'une liste chaînée puis tri :
(représentation des structures de tâche, sans montrer ce qu'elles contiennent)

Schéma du principe :



1er tri suivant les dates de réveil

2 ème étape : toujours exécuter le premier élément de la liste chaînée, puis le replacer au bon endroit.

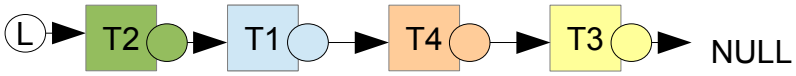


Ici on peut voir la méthode de placement de la tache t4 après son exécution
Pour ce faire nous utilisons des pointeurs :

- csl qui pointe toujours sur l'élément postérieur à la tache analysée
- tmp qui pointe sur l'élément précédant la tache analysée
- elem pointe sur la tache à déplacer.

Ainsi après plusieurs itérations csl est derrière t1 et tmp devant celui ci. Il ne reste qu'à placer cette tache à cet endroit.

Nous obtenons alors :



Gestion des boucles infinies

En temps normal, tout thread créé par notre analyseur doit rendre la main. Ce qui est en temps-réel avec RTAI le rôle de la fonction « rtask_wait() ».

Or il peut arriver que l'étudiant place mal ce code ou l'oublie. La tâche (le thread) s'exécute alors en boucle infinie sans rendre la main.

Notre objectif était donc de sortir de cette boucle infinie après un certain temps éventuellement en la détectant.

Dans un premier temps nous avons regardé les moyens de détecter une boucle infinie. Cela s'avère assez complexe. Nous avons retenu diverses manières pour le faire :

Certification des programmes (exemple : logiciel coq)

La première est de vérifier toutes les boucles une par une pour vérifie qu'elles possèdent toutes une condition de sortir SUR pour empêcher une boucle sans fin.

La deuxième, appelée « *slow-fast* », consistant à exécuter deux programmes en parallèle à raison de deux instructions dans l'un pour une dans l'autre et de comparer à chaque étape les états mémoire : quand on en trouve deux identiques, on est dans une boucle infinie et le programme ne saurait donc se terminer.

Vu la complexité, nous nous sommes donc concentrés sur un simple arrêt de la boucle infinie détectée par détection d'une condition temporelle atteinte.

Les différentes solutions envisagées :

Recherche d'un moyen permettant d'arrêter le thread grace à la libraire pthread.
Mise en place d'un timer.

En premier lieu, nous avons consulter la libraire pthread dont voici quelques fonctions qui nous ont parus utiles :

```
int    pthread_cancel(pthread_t);
```

Fonction permettant de stopper un thread à partir d'un autre.

```
int    pthread_cond_timedwait(pthread_cond_t *,
           pthread_mutex_t *, const struct timespec *);
```

Fonction qui nous a paru utile au premier abord mais qui finalement ne correspondait pas au besoin.

```
int    pthread_setschedparam(pthread_t, int ,
           const struct sched_param *);
```

Fonction qui permet de changer la priorité d'un thread.

Test de ses fonctions :
La première fonction marche comme prévu et stoppe un thread.

Ensuite, nous avons remarqué que pthread_cond_timedwait attendait une condition pour bloquer le

thread. Or nous ne pouvions pas ajouter cette condition dans notre programme, n'ayant pas accès aux fonctions de l'étudiant. Nous avons donc abandonner ce choix.

La 3ème fonction permet d'utiliser la priorité et de fournir un code dont l'optique est la suivante :

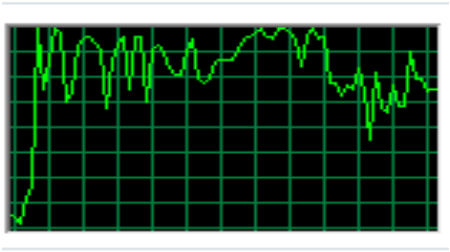
Changer la priorité d'un thread parallèle timer
Lancer le thread du timer et le thread de la tache

Code du timer :

```
void timer()
{
//mettre un délai
sleep(3) ;
// arrêter le thread
pthread_cancel(pthread_t)
}
```

Résultat :

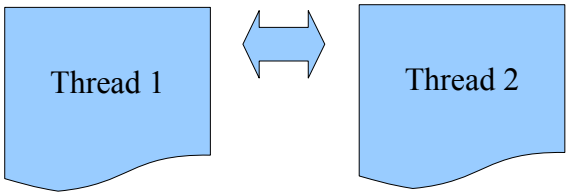
Après quelques tests, nous avons vu que le timer ne stoppait jamais le thread. Nous en avons émis l'hypothèse suivante :le « sleep » attend sûrement une période d'inactivité de 3s pour se terminer. Le programme tournant en boucle infinie, cette période n'a jamais lieu.



Utilisation du cpu réalisée sous code block pour un thread en boucle infinie et un timer.

Nous avons donc envisager d' utiliser une fonction timer qui tourne en boucle infinie.

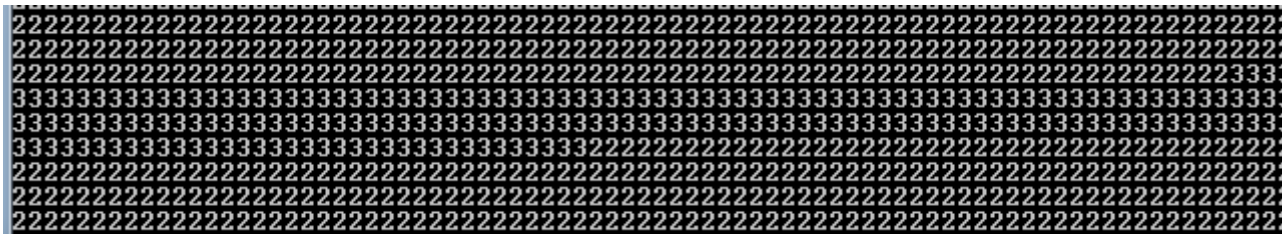
Proposition:



2 thread en exécution boucles infinies.

2 thread de même priorité qui accaparent totalement la charge cpu dédiée au programme.
Ces deux thread s'exécutent donc en parallèle.

Quelques test rapides ont validé notre conception :



Exécution parallèle de deux thread affichant pour l'un 3 et l'autre 2

Objectif : réaliser un simple timer en demandant l'exécution d'incrémentation d'une variable.

Code de la fonction timer :

```
void timer_appli (void)
{
	int umin=0;
	int umax=0;
	    while(umax<TIMER_MAX)
	    {
	        while(umin<100000)
	    {
	      umin++;
	    }
	    umin=0;
	    umax++;
	    if (umax<50000)
	    {
	        printf("value de i:%d\n",umin); //sortie (i);
	    }
	    else{
	    printf("value de i:%d\n",umin);
	        pthread_cancel(tache_courante->thread_tache);}

	    }
	    flag_loop=1;
	    rt_task_wait_period_bis();

}
```

Malheureusement, cette solution ne permet pas un choix précis du temps souhaité mais fonctionne.
Il faut jouer grossièrement sur TIMER_MAX pour avoir un ordre d'idée temporel.

Amélioration de la proposition :

Utilisation de la librairie time.h et de la fonction clock
Ce qui nous a finalement amené à proposer le code suivant :

```
timer()
{
   float temps;
   clock_t t1, t2,t3;

   t1 = clock();

   float temps_timer;
   int u=0 ;

   t3 = clock();
   temps_timer=(float)(t3-t1)/CLOCKS_PER_SEC
   while(temps_timer<TIMEMAX) /
       {
          if(temps_timer<TIMEMAX-2)
          {
            u++;
          }
           else
          { pthread_cancel(thread_boucleinfinie) ;}
          u=0;
          t3= clock();
          temps_timer=(float)(t3-t1)/CLOCKS_PER_SEC;
       }
wait_periodbis() ;
}
```

Bilan après de nombreux tests effectués:

Cette solution est rejetée. En effet elle fonctionne pour un seul thread à la fois et gère bien le temps. En revanche lorsque deux threads sont exécutés simultanément, le compteur de temps ne fonctionne pas et ne génère pas d'arrêts.
Nous avons donc implémenté la solution basique qui gère mal la durée de temps mais effectue le résultat souhaité, à savoir arrêter le boucle infinie.

Défauts de cette méthode :

Si l'exécution de l'analyse est trop longue, le timer arrive à son terme et détecte un problème alors qu'il n'y en a pas. Le principe se base donc sur une exécution « trop longue »
Quelques tests nous ont montrés, qu'avec 1000 cycles le temps d'execution est inférieur à 2s. On peut donc raisonnablement paramétrer le timer sur une durée de 4 à 5 secondes ;

Détection des fautes de segmentation

Dans le cas où une faute de segmentation aurait lieux dans une tâche, c'est à dire que le programme tente de modifier de la mémoire déjà occupée. Ce problème est généralement dût à un problème d'adressage d'une variable.

La librairie signal.h permet d'attraper les signaux, notamment celui indiquant une faute de segmentation. Lorsque ce signal est repéré, la fonction « segfault_sigaction » est alors appelé, ce qui permet de continuer à exécuter le programme.

L'idée est ensuite de pouvoir reprendre l'exécution du programme après l'appel de la fonction qui cause la faute de segmentation. C'est pourquoi nous avons utilisé la librairie « setjmp.h », qui permet par un système d'étiquette de ce rendre à un endroit voulu du programme. Il existe une fonction « goto » pour faire cela mais elle ne permet de ce déplacer qu'à l'intérieur d'une même fonction.

Voici le principe :

On a le code de la tâche contenant potentiellement une faute de segmentation :

```
void code_tache(void)
{
  while(1)
  {
    //code de la tâche
    rt_task_wait_period();
  }
}
```

Puis la fonction qui appel cette tâche, dans laquelle on a placé une « étiquette » pour pouvoir revenir à l'exécution du programme en sautant l'appel de la fonction causant la faute de segmentation :

```
void * demarrage_tache(void *la_tache)
{
  RT_TASK * t = (RT_TASK *) la_tache;

  if ( ! setjmp(buf) )
  {
    // fonction a tester pour le segfault
    t -> nom_fct( t -> arg ); // appel de la tâche
  }
}
```

On a la fonction de capture, dans laquelle on va lors de la détection du signal de faute de segmentation. Celle-ci contient la fonction « longjmp » permettant de se déplacer dans le programme à l'endroit on on a létiquette :

```
void segfault_sigaction(int signal, siginfo_t *si, void *arg)
{
  longjmp(buf,1);
}
```

Il faut également initialiser les fonctions qui permettent de repérer le signal de faute de

```
// initialisation pour attrapper les fautes de segmentation
struct sigaction sa;
memset(&sa, 0, sizeof(sigaction));
sigemptyset(&sa.sa_mask);
sa.sa_sigaction = segfault_sigaction;
sa.sa_flags   = SA_SIGINFO;
sigaction(SIGSEGV, &sa, NULL);
```

Gestion des interruptions

La tâche d'interruption est normalement initialisée avec la fonction « rt_request_global_irq », c'est donc la fonction qu'on utilise pour mémoriser les informations relative à cette tâche. Cette fonction s'active normalement à l'appui sur un bouton et cette tâche est prioritaire. Pour la simulation nous avons choisi de demander à l'utilisateur de renseigner la date d'exécution de cette tâche d'interruption au moment de son initialisation.

On appel ensuite cette tâche comme les tâches périodiques, mais la tâche d'interruption est effectuée en priorité.

Utilisation du Programme de simulation

Exécution du programme de simulation

Le programme à tester doit être placer dans le dossier de simulation avec le programme « simul_TR.c » et le « Makefile ».
Le Makefile doit être modifier pour compiler avec le programme de l'étudiant. Il faut remplacer les endroit surlignés par le nom que l'étudiant a donné au programme C qu'il souhaite analyser :

```
prog_etudiant.o: prog_etudiant.c

        cat prog_etudiant.c
        |sed -e "s/return/return\(0\)/g"
        |sed -e "s/RTIME/int/g"|grep "#include" -v
        |grep "MODULE_LICENSE" -v
        |  sed -e 's/^\(.*rt_task_init\)\(.*,\)\(.*,\)\(.*,\)\(.*,\)\(.*,\)\(.*\)/\1\2\&\3\4\5\6\7\8/'
        > FichierTP.h

        gcc -Wall -o simul_TR simul_TR.c -lpthread -DREENRANT
```

Ensuite il faut lancer le Makefile dans le terminal (ce placer dans le dossier de simulation puis « make »).
(Après une compilation, pour tester un nouveau programme il faut d'abord faire « make clean » pour réinitialiser et pouvoir faire de nouveau une compilation propre).
Ensuite il faut lancer dans le terminal « ./simul_TR » pour lancer le programme de simulation.

Le résultat la simulation se trouve dans le fichier texte « Chronogramme.txt »

Lors de l'exécution de la simulation on demande à l'utilisateur d'entrer :

- Le nombre de cycles.
- La date d'exécution de la tâche d'interruption éventuelle.

Explication du résultat de la simulation

Le programme de simulation donne un résultat de la simulation sous forme de fichier texte :

```
|-------------------------------------------------------------------------------
|                  Programme de Simulation Temps-Réel                          |
-------------------------------------------------------------------------------

|                            | 0 est affiche sur le port 378 du boitier d'E/S
|                            | ouverture module
-------------------------------------------------------------------------------
|                          Declaration des Taches                               |
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
|                        Nouvelle Tache Periodique                              |
-------------------------------------------------------------------------------
|  numero   |  priorite  |  date de reveil (µs)      |  periode (µs)           |
-------------------------------------------------------------------------------
|         1 |          0 |                         0 |                   5000 |
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
|                        Nouvelle Tache Periodique                              |
-------------------------------------------------------------------------------
|  numero   |  priorite  |  date de reveil (µs)      |  periode (µs)           |
-------------------------------------------------------------------------------
|         2 |          1 |                         0 |                   2500 |
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
|                        Nouvelle Tache Periodique                              |
-------------------------------------------------------------------------------
|  numero   |  priorite  |  date de reveil (µs)      |  periode (µs)           |
-------------------------------------------------------------------------------
|         4 |          2 |                         0 |                   5000 |
-------------------------------------------------------------------------------
```

On voit ici le début du programme avec un exemple permettant de montrer les différentes informations relatives à chaque tâche déclarée.
On montre toutes les actions et tout les commentaires du programme étudiant, c'est pourquoi on voit au début « 0 est affiche sur le port 378 du boitier d'E/S » et « ouverture module », ces actions étant effectuées avant les fonctions d'initialisations des tâches.

On peut ensuite voir le résultat de la simulation de l'exécution, montrant à la fin les erreurs éventuelles :

```
-------------------------------------------------------------------------------
|                          Appel des Taches    (2 cycles)                     |
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
| cycle | temps (µs) | tache | action(s) de sortie                           |
-------------------------------------------------------------------------------
|     0 |          0 |     1 |
|                            | 63 est affiche sur le port 378 du boitier d'E/S
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
| cycle | temps (µs) | tache | action(s) de sortie                           |
-------------------------------------------------------------------------------
|     0 |          0 |     3 |
|                            | 1 est affiche sur le port 378 du boitier d'E/S
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
| cycle | temps (µs) | tache | action(s) de sortie                           |
-------------------------------------------------------------------------------
|     0 |          0 |     0 |
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
|                          Fin de la Simulation                               |
-------------------------------------------------------------------------------

ERROR : la tache 1 fait une boucle infinie

ERROR : la tache 1 fait une faute de segmentation

|                            | 127 est affiche sur le port 378 du boitier d'E/S
|                            | fermeture module
```

Limites du programme de simulation

Mode One Shot

Nous n'avons pas eu le temps d'aborder ce mode. Néanmoins, étant donné les bases solides fournies (nombreuses fonctions réécrites qui compilent), on peut ajouter ce mode en effectuant une modification du code.

Exécution multi-modules

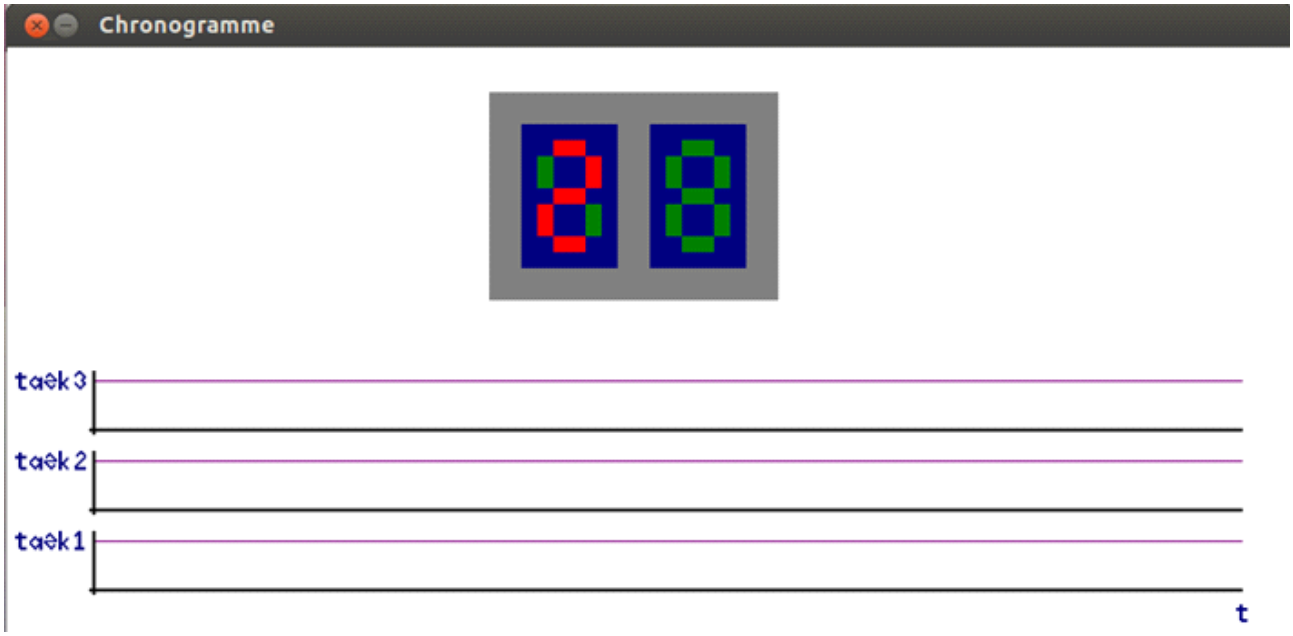
Certaines fonctions de la librairie RTAI offrent la possibilité de transmettre des variables d'un module à l'autre. Cependant cette fonctionnalité ne porte que peu d'intérêt si l'exécution de l'analyseur ne se fait pas en temps réel (attente d'une touche).

Nous aurions aimé mettre en place ce mode mais nous n'avons pas eu le temps de proposer une solution. Nos premières recherches nous ont poussés vers l'utilisation IPC par de la mémoire partagée (processus utilisé par RTAI également semble t-il).

Production graphique

Plusieurs solution s'offraient à nous pour la réalisation graphique, notamment la programmation en langage Python ou la programmation en C avec les librairies GTK+ ou SDL.

Avec la librairie SDL, nous avons réussi à créer une fenêtre sur laquelle nous pouvons afficher des pixels aux endroits souhaités, de la couleur souhaitée. En créant des fonctions pour créer par exemple un rectangle et ainsi de suite nous avons pût arriver au résultat suivant :



Il faudrait ensuite gérer l'affichage de l'échelle de temps et placer les dates de réveil des tâches au bon endroit, mais le résultat graphique n'étant pas une priorité pour la maîtrise d'ouvrage, nous nous sommes arrêté là.

Expérience acquise et Bilan

Nouvelles compétences en programmation

Ce projet nous a permis d'obtenir d'améliorer nos compétences en langage C sur :

- l'utilisation de fonctions Callback,
- la gestion d l'ordonnancement avec des listes chaînées,
- la gestion de threads (création, manipulation, priorité , mutex, etc...),
- le multi threading,
- l'utilisation des librairies pthread, signal, setjmp et SDL.

Nous avons aussi aussi fait appel à du shell Linux pour la compilation.
Nous avons découvert des notions d'autre langages dans nos recherches, par exemple avec la manipulation de thread en JAVA(runtime) et C++. Pour détecter les erreurs de segmentations nous avons aussi essayé de passer en C++, mais cela causait d'autres problèmes.
Nous nous sommes également penché sur l'étude du langage Python pour la partie graphique.

Expérience personnelle

Ce projet fût un défi. Il demandait une organisation autonome, tant dans la gestion du temps que dans la mise en place des solutions.

Il fallait d'abord être d'accord avec l'encadrant sur le cahier des charges et ses objectifs.

Les problèmes à résoudre demandaient d'aller au delà de nos connaissances acquises en cours. Nous avons alors dût rechercher et comprendre par nous même les solutions éventuelles.

Nous avons aussi dût nous adapter en équipe pour nous séparer le travail sans perdre de vue l'objectif final. Il fallait toujours être d'accord pour essayer d'être complémentaire au maximum. Il fallait être clair lors de la mise en commun.

Nous avons également pût voir l'importance de l'entente dans l'équipe, tant pour produire des solution cohérentes, tenant compte du travail d'autrui, que pour une motivation générale de l'équipe.

Bilan

Ce projet de fin d'année d'IMA 4 aboutit à une solution exploitable qui répond aux principales attentes du cahier des charges. Cette solution permet en effet de visualiser plus en détail l'exécution des tâches temps-réelles et de signaler les principales erreurs inductrices de problèmes lors des travaux pratiques des cours de temps-réel.

Ce projet nous à montré l'importance de la gestion du temps et l'évaluation correcte du temps nécessaire à la mise en place de chaque solution. Il reste des améliorations à apporter à ce projet, il peut être ouvert de nouveau pour permettre de simuler plus de fonctions temps-réels et produire une partie graphique plus ergonomique pour les utilisateurs.

D'un point de vue interne, ce fût dans l'ensemble pour notre équipe une occasion qui nous a permis de progresser sur la gestion de projet, l'entente en équipe, l'autonomie au niveau de la réalisation et la remise en question des méthodes et des solutions apportées.

Remerciements

Nous tenons à remercier monsieur Blaise Conrard, qui nous a aidé autant que faire ce peut tout en nous laissant une grande part d'autonomie. Ces propositions lors du projet nous ont permis d'apprendre de nombreuses connaissances. Il nous a également débloqué lorsque nous en avions besoin et nous le remercions vivement pour cette aide précieuse.

Nous remercions également monsieur monsieur Julien Forget pour le temps qu'il nous a accordé, pour ses indications ainsi que ses conseils adaptés.

Nous remercions de plus les membres de l'équipe pour le travail accompli et l'entente qu'ils ont su garder.

Nous remercions enfin Xavier Redon pour le travail qu'il fournit à gérer l'organisation et l'encadrement global des projets.

Annexe :

Tableau récapitulatif des actions effectuées triées par catégorie :

<table>
<tr><td colspan="2">Création Analyseur:</td></tr>
<tr><td>Utilisation commandes bash pour compiler</td><td>Création structures + listes chaînées adaptées.</td></tr>
<tr><td>Réécriture des fonctions RTAI</td><td></td></tr>
<tr><td colspan="2">Gestion affichage fichier log :</td></tr>
<tr><td colspan="2">Création fichier + mise en forme</td></tr>
<tr><td colspan="2">Ordonnancement:</td></tr>
<tr><td>V1 :Ordonnancement parcours global</td><td>V2 :Ordonnancement 1er élément à exécuter.</td></tr>
<tr><td>Utilisation thread.</td><td>Utilisation de mutex.</td></tr>
<tr><td colspan="2">Test boucle infinie :</td></tr>
<tr><td>Test timer : cond_timedwait(), gettimeoftheday() etc...</td><td>Test changement de priorité+ thread libre et reprise en main processeur.</td></tr>
<tr><td>Recherche solutions C++.</td><td>Test timer par méthode boucle infinie</td></tr>
<tr><td colspan="2">Test segfault :</td></tr>
<tr><td>Utilisation librairie signal
Utilisation librairie jump</td><td>Recherche solutions C++ (try-catch)</td></tr>
<tr><td colspan="2">Gestion simulation d'interruption :</td></tr>
<tr><td colspan="2">Réécriture fonction d'interruption et code adapté</td></tr>
<tr><td colspan="2">Test Partie graphique :</td></tr>
<tr><td>SDL + programme d'essai</td><td>Graphique en Python</td></tr>
<tr><td colspan="2">Solution passage en C++ :</td></tr>
<tr><td>Changement quelques fonctions</td><td>Blocage utilisation callback en C++</td></tr>
</table>