Auteur : Moyart Yann

Diplôme : Master

Jalon intermediaire : PJE

Date : 15 Janvier 2015 – Juin 2016

Responsable : Giuseppe Lipari

Sujet du projet :

*Support à la programmation temps-réel sur Linux*

Sommaire

[Introduction au projet : 3](#_Toc445114339)

[Objectif : 4](#_Toc445114340)

[Planification : 5](#_Toc445114341)

[Réalisation du projet : 6](#_Toc445114342)

[Etude de la librairie PTASK. 6](#_Toc445114343)

[Interprétation du problème 6](#_Toc445114344)

[Réalisation d’un exemple de déplacement d’échéance : 8](#_Toc445114345)

[Conception d’une interface : 9](#_Toc445114346)

[Réalisation de tests 9](#_Toc445114347)

[Problèmes Rencontrés 10](#_Toc445114348)

[Résultats 10](#_Toc445114349)

[Bilan 10](#_Toc445114350)

[Bilan personnel 10](#_Toc445114351)

[Bilan Professionnel 10](#_Toc445114352)

[Index 10](#_Toc445114353)

[Conclusion 10](#_Toc445114354)

# Introduction au projet :

Les systèmes temps réel sont des systèmes dont le bon fonctionnement repose sur l'exactitude des résultats produits aussi bien que sur le respect des contraintes temporelles. Un système temps réel est souvent implanté par un ensemble de processus concurrents qui s'exécute sur un système d'exploitation. Les développeurs ont besoin de spécifier des contraintes temporelles et de vérifier qu'elles seront respectées pendant l'exécution du logiciel. Notamment, il est nécessaire de vérifier que le délai d'exécution d'un processus est borné par une échéance.

Un système en temps réel est celui où la rapidité du résultat de calcul est importante. Les exemples incluent les systèmes d'armes militaires, des systèmes de contrôle et le streaming vidéo et audio.

# Objectif :

Etudier et Utiliser la librairie PTASK pour créer des tâches périodiques sous Linux dans un second temps étendre la librairie PTASK en ajoutant des fonctions permettant de détecter les fautes temporelles en particulier détecter l’échéance d’une tache périodique.

# Planification :

* Etude des librairies associés (temps estimé : 1 semaine)
* Réalisation d’un exemple de déplacement (temps estimé : 2 semaine)
* Conception d’une interface pour ajouter ce système dans la librairie PTASK (temps estimé 1 – 2 jours)
* Réalisation de tests automatiques pour vérifier le fonctionnement du mécanisme (temps estimé 2 – 3 semaines)

# Réalisation du projet :

## Etude de la librairie PTASK.

La première étape fut de me familiariser avec la librairie PTASK indispensable pour l’avancement du projet, j’ai donc téléchargé la librairie depuis le dépôt Git hub.

La librairie PTASK est une extension de la librairie POSIX standard real-time. POSIX RT est conçu selon plusieurs caractéristiques importantes :

* *Timers* : minuteries périodiques, leurs fonctionnements utilisent des signaux POSIX
* *Ordonnancement de priorité* : Ordonnancement préemptif avec un minimum de 32 niveaux de priorité.
* *Signaux en temps réel* : signaux supplémentaires avec de multiples niveaux de priorité
* *Sémaphores*.
* *Les files d'attente de mémoire*: passage de messages utilisant les files d'attente nommées.
* *Mémoire partagée*: les régions de mémoire nommés partagés entre plusieurs processus.
* *Verrouillage de la mémoire*: les fonctions pour empêcher la permutation de la mémoire virtuelle avec les pages de mémoire physique.

On trouvera dans la librairie PTASK : un dossier « /src » contenant les sources de la librairie, un dossier « /test » contenant les différentes modifications apportés aux interfaces,  « /examples » contenant différents exemples d’utilisation, notamment des programmes utilisant l’interface graphique allegro. Un dossier « /docs » contenant la documentation de la librairie.

### Interprétation du problème

Dans l’idée on enregistre une période et une date d’échéance dans un processus, durant la période le processus peut s’exécuter. Ce temps d’exécution peut prendre plus ou moins de temps en fonction des autres taches en cours, et dans le cas où le temps d’exécution de ce processus atteint la date d’échéance, c’est-à-dire la fin de la période qui lui est impartis, il continuera son exécution a la période suivante.

Temps d’exécution

Début du processus

Echéance du processus

Période du processus

Echéance dépassé

Temps d’exécution

*Schématisation du problème général. Exécution d’un processus, sans dépassement à gauche et avec dépassement à droite.*

La solution serait de vérifier quand un processus atteint son échéance et stopper son exécution grâce aux signaux de la librairie POSIX.

Temps d’exécution

Début du processus

Echéance du processus

Période du processus

Signal d’interruption

Temps d’exécution interrompu

*Schématisation de la solution. Exécution d’un processus, sans dépassement à gauche et avec dépassement à droite, on corrige le problème en intégrant l’arrivé d’un signal INT ou KILL.*

## Réalisation d’un exemple de déplacement d’échéance :

Après avoir étudié la librairie, imprimé la documentation et testé les quelques exemples d’utilisation. Il fallait que je créer un processus avec un temps d’exécution délimité par une échéance.

J’ai donc étudié la librairie POSIX un peu plus en détails sur les timers et les sigaction.

L’objectif était de créer un timer qui s’incrémente toutes les x millisecondes, et arrivé à l’échéance du processus il le stoppe.

**Processus**

Pid

Echeance

**Timer**

Interval

**Timer Handler**

Kill

Dans cette schématisation de l’exemple, le timer et le processus sont 2 entités reliées à un Handler. A chaque tic du timer le Handler sera interrogé :

*If( temps\_execution(pid) > échéance (pid))*

*Then kill (pid)*

*Else*

*Relance le timer*

Cet exemple simple montre le fonctionnement global du problème. Il faut à présent généraliser la solution, dans le cas où il y a plusieurs processus en cours.

## 

## Conception d’une interface :

Cette étape consiste à intégrer le système à la librairie PTASK.

Etendre cette solution à la librairie signifie ajouter  des méthodes à des nouveau modules sans toucher aux modules existant, sans recompiler ou modifier les sources.

J’ai donc créée un module « deadline\_handler.c » et « deadline\_handler.h »

## Réalisation de tests

Une fois la solution du handler mise en place il faut trouver tester plusieurs cas :

* Chaque processus s’exécute correctement dans leurs périodes imparties.
* Un ou plusieurs processus dépasse l’échéance car le «calcul » effectué par le processeur est trop long.

Pour réaliser ces tests j’ai mis en place un programme qui créer 3 processus liée à 3 timers, chacun d’eux exécute séparément une suite de fibonacci en suivant le fonctionnement général de ma solution, c’est-à-dire :

1. Lancer le timer
2. Je lance 10 périodes de 2 seconde pour
   1. Exécution d’une fonction lambda dans mon cas : Afficher le résultat de fibonacci (x)
   2. Relancer le timer
   3. Attendre la fin de la période
3. Supprimer le timer

### Observations

Aujourd’hui j’obtiens les résultats suivants :

* Pour Fibonacci(10)

1. Creation des taches
2. Un processus m’indique le résultat de fibonacci(10)
3. Reset du timer de ce processus
4. Répétition de 2 et 3
5. Suppression du timer et terminaison de son processus lorsqu’il a affiché 10 fois le résultat

* Pour Fibonacci(40)

1. Création des taches
2. Détection d’un dépassement du processus 0
3. Arrêt du processus 0 Suppression du timer liée à ce processus
4. Détection d’un dépassement du processus 1
5. Arrêt du processus 1 Suppression du timer liée à ce processus
6. Détection d’un dépassement du processus 2
7. Arrêt du processus 2 Suppression du timer liée à ce processus
8. Détection d’un dépassement du processus 3
9. Arrêt du processus 3 Suppression du timer liée à ce processus

* Pour Fibonacci(35)

1. Création des taches
2. Un processus m’indique le résultat de fibonacci(35)
3. Reset du timer de ce processus
4. Répétions de 2 et 3
5. Détection d’un dépassement du processus 1
6. Arrêt du processus 1 Suppression du timer liée à ce processus
7. Terminaison des processus 2 et 3 lorsqu’ils ont affiché 10 fois le resultats de la fonction

Ce cas est un cas aléatoire puisque le processus 1 s’exécute 1, 2, 3, parfois 4 fois avant de détecter un dépassement.

### Interprétation

Pour Fibonacci(10) le calcul est fait plus rapidement par le processeur, la fonction se termine donc avant l’échéance du processus, tous les processus se terminent correctement après avoir fait leur calcul.

P3

P2

P1

P3

P1

Début du processus

Echéance du processus

Période du processus

P2

*Shematisation de l’interprétation de fibonacci(10)*

Pour Fibonacci(40) le calcul est trop long pour la période prédéfinie, chaque timer de chaque processus détecte le dépassement et stoppe instantanément le processus dès que l’échéance est atteinte tout en supprimant le timer associé.

P2

P3

P1

Début du processus

Période du processus

Signal d’interruption

P1

P1

P2

P3

*Schématisation de l’interprétation de fibonacci(40)*

Pour Fibonacci(35) on sait que le processus 1 s’exécute un certain nombre de fois en fonction des exécutions du test. Le processeur ne peut pas exécuter autant de fois cette fonction en 2 seconde un des processus met donc plus de temps que les autres à s’exécuter.

Signal d’interruption

P3

P2

P1

P3

P1

Début du processus

Echéance du processus

Période du processus

P2

P1

P2

P3

*Schématisation de l’interprétation de fibonacci(35)*

# Problèmes Rencontrés

# Résultats

# Bilan

## Bilan personnel

## Bilan Professionnel

# Index

# Conclusion