Projet Xenomai - Temps réel

Walid GHETTAS

Encadrant: Pierre FICHEUX

Année universitaire 2020 - 2021

Au cours de ma dernière année d'études d'ingénieur, j'ai eu un enseignement sur les systèmes dits temps réel. À l'issue des enseignements à la fois théorique et pratique, un projet m'a été imposé. En utilisant les exemples Xenomai étudiés en cours, je devais créer une application me permettant de faire varier la période du thread TR Xenomai depuis le shell Linux. Ainsi, à partir d'un programme fourni par l'encadrant, j'ai dû le compléter afin d'être capable de modifier les périodes d'un thread et ce, depuis un autre terminal. Une fois finalisé l'écriture du programme, j'ai cross-compilé pour pouvoir exécuter ce programme sur une RaspBerry Pi 3 Model B et j'ai observé les résultats sur le terminal. Tout au long de ce rapport, je prendrai le temps de détailler chacune des étapes faites pour mener à bien ce projet.

Avant toute chose, il faut savoir que j'avais à ma disposition un câble Ethernet et un câble TTL en plus d'un kit basique d'une Raspberry Pi composé d'un câble d'alimentation, carte mémoire de type micro SD et son adaptateur ainsi que la Raspberry Pi elle-même. Je me suis assuré que le branchement était réalisé en bonne et due forme avant de me lancer dans l'étude du code qui m'avait été fourni par l'encadrant.

Le programme était déjà bien rempli. Tout ce que j'ai eu à effectuer a été d'utiliser la fonction *realtime_thread(void *arg)* en m'aidant d'un exemple d'utilisation du protocole XDDP qui était disponible dans le répertoire *XDDP/* des TP Xenomai. J'ai également ajouté la bibliothèque *rtdm/ipc.h* ainsi qu'adapté pour la création d'un thread supplémentaire dans le main. Une dernière chose importante avant de cross-compiler a été de prendre en compte le changement de *period_ns* lors de l'entrée de la commande *echo*. En s'assurant que *period_ns* est bien une variable globale, je me suis permis de faire ce changement à la fin de la seule fonction qui a été ajoutée à ce programme.

```
209       if (ret <= 0)
210         perror("recvfrom");
211       else{
212         printf("received <%s> (%d)\n", buf, ret);
213         period_ns/=atoi(buf);
214       }
215     }
216   }
217
```

La ligne 213 est la ligne clé permettant le changement de période lorsqu'on fait varier la période du thread TR Xenomai depuis un shell Linux.

Une fois l'écriture du code finalisé, j'ai effectué une cross-compilation. En effet, cette étape est nécessaire puisque celle-ci permet de générer sur une architecture X un exécutable qui sera compatible avec une architecture Y. Dans mon cas, j'ai compilé depuis mon ordinateur personnel composé d'une architecture x86-64 afin de générer des

exécutables compatibles avec la Raspberry Pi 3, qui elle est équipée d'une architecture ARM. Différentes commandes, listées ci-dessous, ont été nécessaires pour passer outre cette étape:

source /opt/poky/3.0.2/environnement-setup-cortexa7t2hf-neon-vfpv4-poky-linux-gnueabi
source ~/bin/set_env_xeno_sdk_Yocto.sh
make

Une fois cross-compilé, nous avons la confirmation que le code ne comporte pas d'erreurs lexicales ou syntaxiques. À l'aide de la commande scp, j'ai alors copié le fichier exécutable vers ma Raspberry Pi. Pour m'assurer du bon fonctionnement de la copie, j'ai préalablement ouvert un autre terminal et me suis connecté en SSH sur ma RaspBerry Pi. J'ai ouvert un dossier intitulé Projet, dossier où je comptais mettre les fichiers relatifs à ce projet. J'ai ainsi pu réaliser la copie par le biais de la commande ci-dessous (en veillant toutefois à être dans le répertoire de l'exécutable):

scp xenomai_rpi_gpio root@10.42.0.81:/home/root/Projet/

Ainsi, après cette dernière commande, le fichier est prêt à être exécuté sur la RaspBerry Pi avec la commande:

./xenomai_rpi_gpio –p <période-en-ns> -g <numéro-gpio>

Après l'exécution du programme, nous lançons en parallèle un autre terminal en SSH. C'est à partir de ce dernier que nous pourrons modifier la valeur de la période initiale. Voici un exemple de commande que nous pouvons lancer dans le nouveau terminal pour modifier la période:

echo 5 > /dev/rtp0

Ici, la période est divisée par 5. Par exemple, si *period_ns* vaut initialement 10, à la suite de la commande *echo*, *period_ns* vaudra 2: l'encadrant ayant spécifié que nous devions prendre en compte uniquement le cas du diviseur.

J'ai personnellement exécuté le code de la façon suivante:

./xenomai_rpi_gpio –p 10000000 -g 40

```
root@raspberrypi3:~/Projet# ./xenomai_rpi_gpio -p 10000000 -g 4
0
Using GPIO 40 and period 10000000 ns
Waiting for XDDP data...
Loop= 200 sec= 1623 nsec= 129009537 delta= 10000000 ns jitter cur= 0 ns avg= 9581810 ns max= 2656 ns
Loop= 400 sec= 1625 nsec= 129009432 delta= 9999636 ns jitter cur= 364 ns avg= 77 ns max= 2656 ns
```

Au lancement, nous observons un delta relativement proche de la période initialement indiqué, 10 millions. Observons la valeur du jitter qui prend en compte la différence en nanosecondes entre la période du thread TR Xenomai que l'on et celle que l'on veut (ici 10 millions). Elle vaut à la première reprise 0 (c'est parfaitement ce que l'on veut). Par la suite, elle est certes différente mais reste minime (assez pour que le système soit temps réel. Rappelons qu'un système temps réel est appelé tel quel, si la valeur de la réponse attendue est plus rapide qu'une valeur que nous aurions considéré comme critique en amont). En parallèle, sur l'autre terminal en SSH, j'envoie successivement les commandes suivantes.

```
root@raspberrypi3:~/Projet# echo 2 > /dev/rtp0
root@raspberrypi3:~/Projet# echo 5 > /dev/rtp0
root@raspberrypi3:~/Projet# echo 3 > /dev/rtp0
```

Normalement, la période devrait respectivement être divisée par 2, 5 et enfin 3. Cela veut dire qu'elle devrait successivement valoir 5 millions, 1 million puis environ 333333. Au même rythme que nous avons entré les commandes ci-dessus, nous avons pu observé sur le premier terminal en SSH les résultats suivants:

```
received <2
> (2)
Waiting for XDDP data...
Loop= 600 sec= 1627 nsec= 74009483 delta= 5000000 ns jitter cur= 0 ns avg= 25197 ns max= 5004323 ns
Loop= 800 sec= 1628 nsec= 74009587 delta= 5000156 ns jitter cur= 156 ns avg= 75 ns max= 5004323 ns
Loop= 1000 sec= 1629 nsec= 74009430 delta= 5000052 ns jitter cur= 52 ns avg= 68 ns max= 5004323 ns
Loop= 1200 sec= 1630 nsec= 74009482 delta= 5000052 ns jitter cur= 52 ns avg= 73 ns max= 5004323 ns
Loop= 1400 sec= 1631 nsec= 74009429 delta= 4999948 ns jitter cur= 52 ns avg= 88 ns max= 5004323 ns
Loop= 1600 sec= 1632 nsec= 74009533 delta= 5000104 ns jitter cur= 104 ns avg= 91 ns max= 5004323 ns
```

Cette première capture d'écran montre les résultats que nous avons pu observer immédiatement après la commande **echo 2 > /dev/rtp0**. Nous pouvons voir que la valeur du jitter reste minime et que la valeur du delta (qui est la période du thread TR Xenomai) a été modifiée. Elle est d'environ 5 millions comme précédemment énoncé. La valeur max a également son importante. Lors de test en industriel, c'est d'ailleurs cette valeur prédomine sur les autres. Les valeurs restent plutôt fidèles à la période estimée. Nous lançons ensuite la commande **echo 5 > /dev/rtp0**.

```
received <5
> (2)
Waiting for XDDP data...
Loop= 1800 sec= 1633 nsec= 18009481 delta= 1000000 ns jitter cur= 0 ns avg= 20210 ns max= 5004323 ns
Loop= 2000 sec= 1633 nsec= 218009481 delta= 999948 ns jitter cur= 52 ns avg= 124 ns max= 5004323 ns
Loop= 2200 sec= 1633 nsec= 418009272 delta= 999739 ns jitter cur= 261 ns avg= 96 ns max= 5004323 ns
Loop= 2400 sec= 1633 nsec= 618009533 delta= 1000053 ns jitter cur= 53 ns avg= 92 ns max= 5004323 ns
Loop= 2600 sec= 1633 nsec= 818009480 delta= 999948 ns jitter cur= 52 ns avg= 70 ns max= 5004323 ns
Loop= 2800 sec= 1634 nsec= 18009480 delta= 1000000 ns jitter cur= 0 ns avg= 102 ns max= 5004323 ns
Loop= 3000 sec= 1634 nsec= 218009480 delta= 1000000 ns jitter cur= 0 ns avg= 96 ns max= 5004323 ns
Loop= 3200 sec= 1634 nsec= 418009376 delta= 1000000 ns jitter cur= 0 ns avg= 73 ns max= 5004323 ns
Loop= 3400 sec= 1634 nsec= 618009376 delta= 999896 ns jitter cur= 104 ns avg= 84 ns max= 5004323 ns
Loop= 3600 sec= 1634 nsec= 818009480 delta= 1000000 ns jitter cur= 0 ns avg= 95 ns max= 5004323 ns
Loop= 3800 sec= 1635 nsec= 18009428 delta= 1000052 ns jitter cur= 52 ns avg= 79 ns max= 5004323 ns
Loop= 4000 sec= 1635 nsec= 218009480 delta= 1000052 ns jitter cur= 52 ns avg= 105 ns max= 5004323 ns
Loop= 4200 sec= 1635 nsec= 418009428 delta= 1000052 ns jitter cur= 52 ns avg= 103 ns max= 5004323 ns
Loop= 4400 sec= 1635 nsec= 618009480 delta= 1000000 ns jitter cur= 0 ns avg= 105 ns max= 5004323 ns
Loop= 4600 sec= 1635 nsec= 818009428 delta= 1000000 ns jitter cur= 0 ns avg= 77 ns max= 5004323 ns
Loop= 4800 sec= 1636 nsec= 18009532 delta= 1000261 ns jitter cur= 261 ns avg= 99 ns max= 5004323 ns
Loop= 5000 sec= 1636 nsec= 218009427 delta= 1000052 ns jitter cur= 52 ns avg= 73 ns max= 5004323 ns
```

Nous avons estimé précédemment que la période devait être de 1 million de nanosecondes. Le delta s'est bien équilibré à nos attentes et la valeur du jitter reste faible, ce qui est satisfaisant pour le moment.

```
received <3
> (2)
Waiting for XDDP data...
Loop= 5200 sec= 1636 nsec= 412676198 delta= 333385 ns jitter cur= 52 ns avg= 3456 ns max= 5004323 ns
Loop= 5400 sec= 1636 nsec= 479342656 delta= 333281 ns jitter cur= 52 ns avg= 112 ns max= 5004323 ns
Loop= 5600 sec= 1636 nsec= 546009323 delta= 333437 ns jitter cur= 104 ns avg= 89 ns max= 5004323 ns
Loop= 5800 sec= 1636 nsec= 612675938 delta= 333334 ns jitter cur= 1 ns avg= 77 ns max= 5004323 ns
Loop= 6000 sec= 1636 nsec= 679342500 delta= 333333 ns jitter cur= 0 ns avg= 91 ns max= 5004323 ns
Loop= 6200 sec= 1636 nsec= 746009063 delta= 333282 ns jitter cur= 51 ns avg= 77 ns max= 5004323 ns
Loop= 6400 sec= 1636 nsec= 812675625 delta= 333333 ns jitter cur= 0 ns avg= 82 ns max= 5004323 ns
Loop= 6600 sec= 1636 nsec= 879342344 delta= 333281 ns jitter cur= 52 ns avg= 83 ns max= 5004323 ns
Loop= 6800 sec= 1636 nsec= 946008854 delta= 333281 ns jitter cur= 52 ns avg= 82 ns max= 5004323 ns
Loop= 7000 sec= 1637 nsec= 12675573 delta= 333333 ns jitter cur= 0 ns avg= 74 ns max= 5004323 ns
Loop= 7200 sec= 1637 nsec= 79342083 delta= 333385 ns jitter cur= 52 ns avg= 82 ns max= 5004323 ns
Loop= 7400 sec= 1637 nsec= 146008698 delta= 333229 ns jitter cur= 104 ns avg= 71 ns max= 5004323 ns
Loop= 7600 sec= 1637 nsec= 212675260 delta= 333385 ns jitter cur= 52 ns avg= 70 ns max= 5004323 ns
Loop= 7800 sec= 1637 nsec= 279341875 delta= 333386 ns jitter cur= 53 ns avg= 79 ns max= 5004323 ns
```

Suite à la dernière commande **echo 3 > /dev/rtp0**, nous pouvons voir que la période a encore été modifiée comme nous l'espérions. À chaque changement de la valeur de la période du thread TR Xenomai effectué via la commande *echo* depuis un autre terminal du shell Linux, nous avons toujours eu des résultats satisfaisants. Le programme pouvait être considéré comme répondant aux consignes du projet.

Pour conclure, la réalisation de ce projet a été un succès. En effet, le but du projet était d'utiliser les exemples Xenomai étudiés en cours afin de créer une application donnant la possibilité de faire varier la période du thread TR Xenomai depuis le shell Linux en utilisant XDDP. Dans la dernière section, nous avons pu observer que les résultats étaient fidèles aux attentes que nous avions. Ce projet m'a permis de prendre en main l'outil Xenomai, et de bien comprendre en quoi cette une extension libre du noyau Linux lui apportant des fonctionnalités temps réel.

FIN