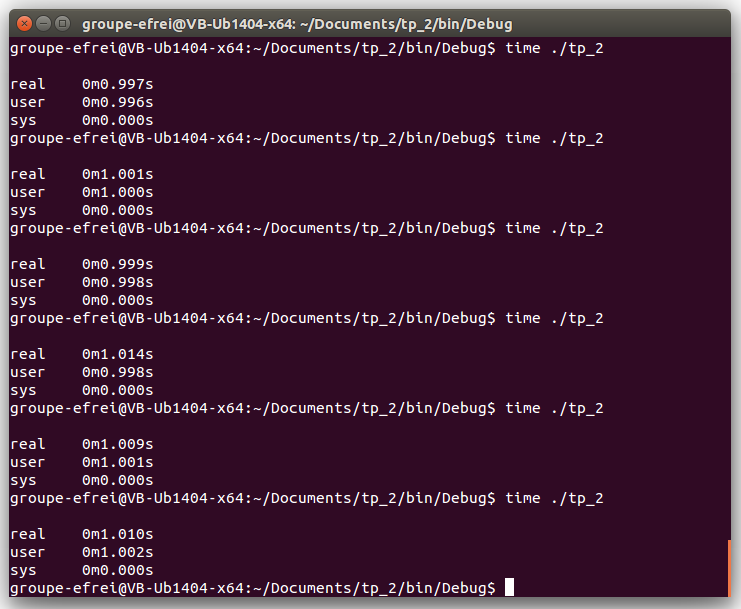
# Rapport Real-Time Systems: TP2

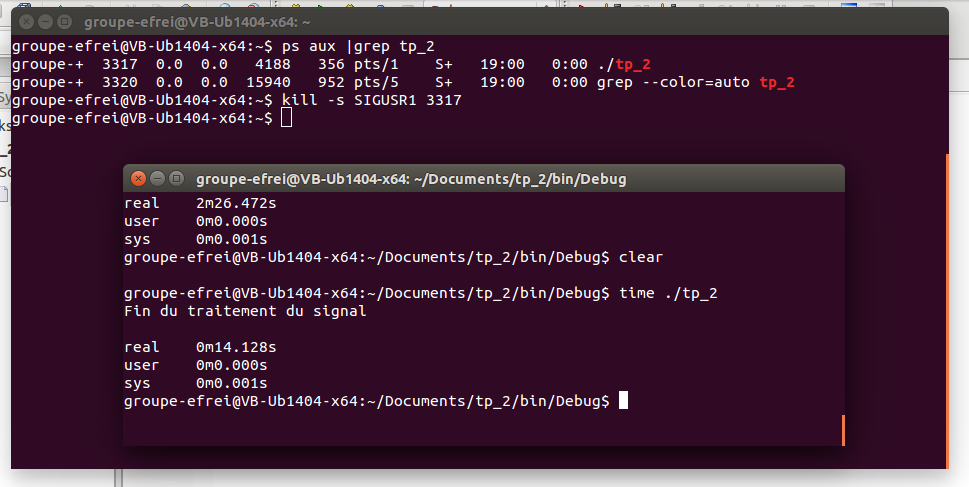
## Question 1



Multiples exécution de la fonction do\_work() avec un temps de travail de 1000 millisecondes

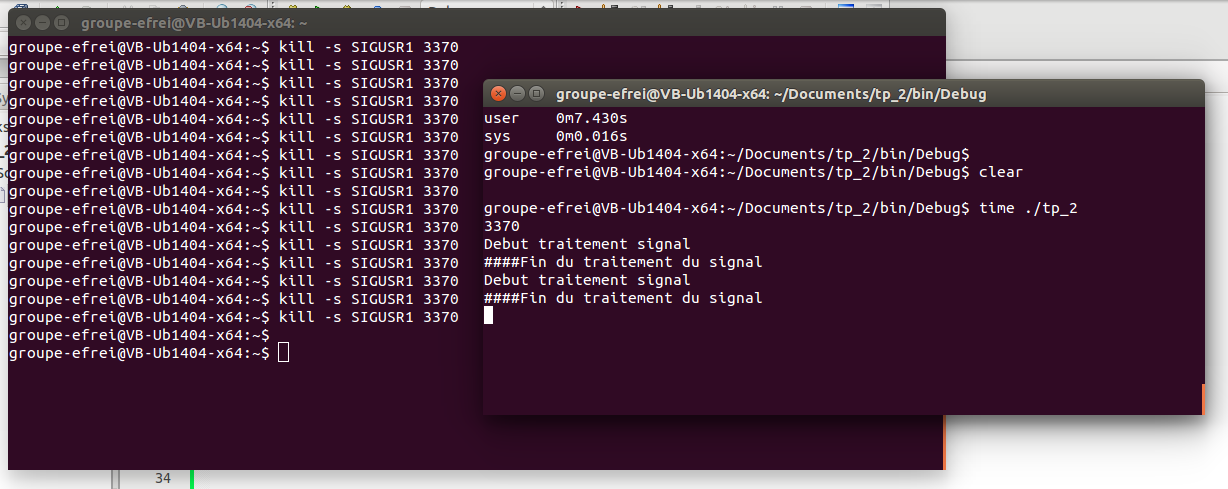
Après avoir exécuté cette fonction plusieurs fois, on peut voir que le temps réel d’exécution de notre fonction n’est pas exact et varie d’environ 10 millisecondes par rapport à la valeur théorique.

## Question 2



Lancement du programme et envoie d’un signal SIGURS1 via la commande kill

## Question 3

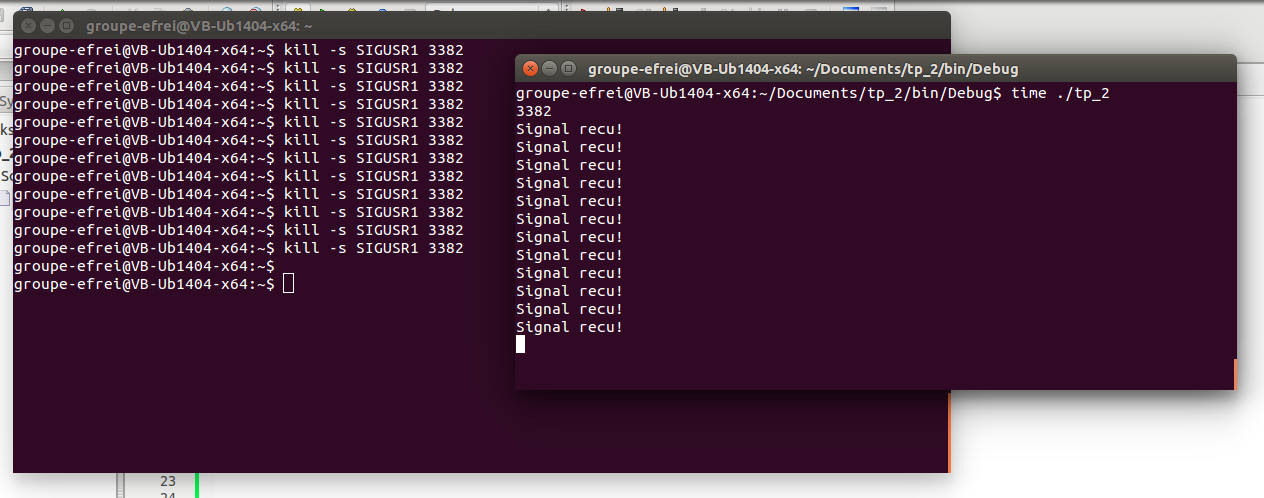


Lancement de 16 signaux et résultats renvoyés par le programme

On intègre la fonction do\_work (cf. q1) dans notre fonction signal handler de manière à rendre son travail plus conséquent et alors nous essayons d’envoyer une rafale de signal et à partir de ce moment nous observons que nous ne récupérons pas tous les signaux émis. Ils ne sont pas tous captés. Le souci est lié au fait que lorsque que le programme est entrain de traiter un signal il ne peut capter un autre signal.

La raison pour laquelle tous les signaux ne sont pas traités c’est que lors d’un traitement les multiples interruptions reçus ne sont pas comptabilisées. On n’en compte qu’une seule en réalité. De ce fait toutes les interruptions arrivant durant un traitement ne seront comptabilisées comme une seule et unique, d’où les pertes de signaux.

## Question 4

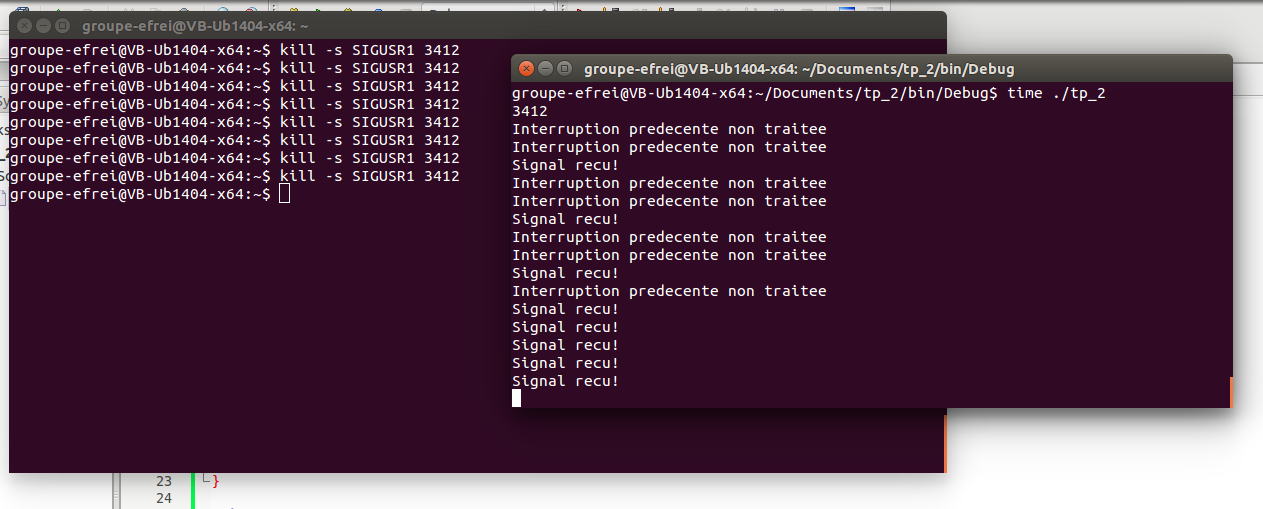


Envoie de 12 signaux et réception correcte des 12 signaux par le programme

Pour réduire la perte il nous faut chercher à rendre le plus disponible possible le handler pour l’écoute d’entrée de signaux tout en autorisant les traitements. En séparant le traitement de l’écoute cela nous permet de rendre bien plus disponible notre processus pour l’entrée d’un signal.

Ce découpage semble alors parfait et nous permettre d’intercepter toutes les interruptions. Mais ce n’est pourtant toujours pas le cas car le gestionnaire d’interruption à toujours un temps d’exécution qui est non nulle et supérieur à zéro. De ce fait si l’on observe deux interruptions sur un temps ou le gestionnaire d’interruption a déjà intercepté un signal alors il ne pourra pas le traiter.

## Question 5



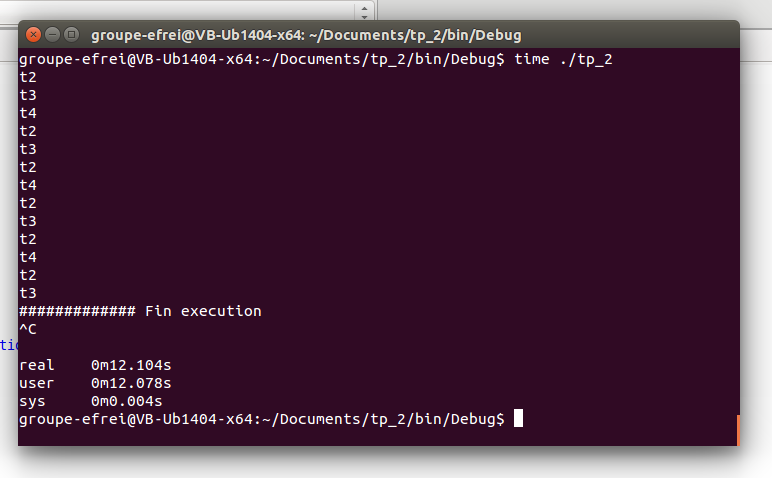
Envoie de 8 signaux

Pour vérifier que l’interruption précédente a bien été traitée quand on reçoit la suivante, on met en place 2 compteurs. Le premier compteur est incrémenté quand un signal est reçu et le second compteur est incrémenté quand le signal est traité. Si jamais la différence entre le premier et le second compteur est supérieure à 1, alors c’est que toutes les interruptions n’ont pas été traitées.

## Question 6

L’inconvénient de l’utilisation de alarm() est que son utilisation est combinée à pause(), le programme devient donc bloquant. De même sleep() rend le programme bloquant en réalisant le même travail que les fonctions alarm() et pause() réunies.

## Question 9



Exécution du programme avec un processeur 2 fois plus rapide : 12 secondes d’exécution au lieu de 6

Lorsqu'on augmente la vitesse d'exécution du processeur, notre programme ne peut plus fonctionner correctement. En effet, le temps de travail est totalement dépendant de la vitesse du processeur (plus le processeur est rapide plus le travail sera cours) tandis que la durée des périodes pour chaque tâche est indépendante et doit rester la même quelle que soit la vitesse. L’ordonnancement élaboré à la question précédente et tenant exactement sur un cycle de 12 secondes sans pause s’exécute désormais en 6 secondes.

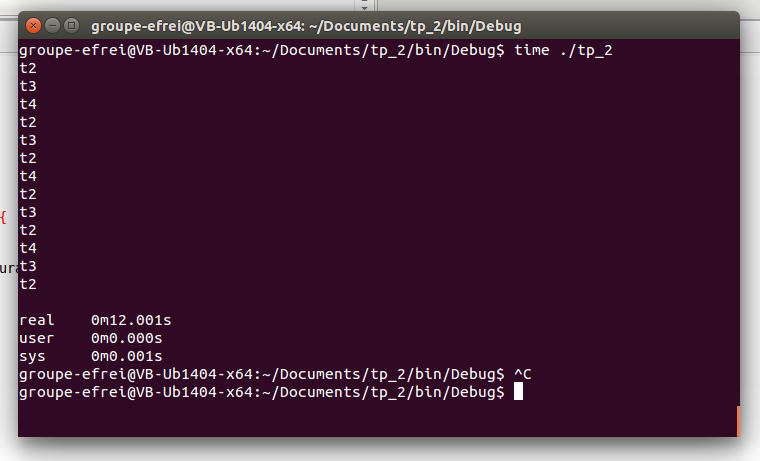
Pour résoudre le problème, il faut insérer entre chaque travail des pauses dont le temps devra être calculé indépendamment de la vitesse du processeur et donc empêcher l'exécution d'une tâche avant que ce temps d'attente ait été exécuté

Si mon processeur est plus que 5 fois plus rapide, je commence à avoir des temps de calcul inférieurs à 12 secondes (11 secondes environ).

## Question 10

L'exécution est bien déterministe, car l'ordre des tâches exécutées est toujours défini dans le programme de manière séquentielle.

## Question 11



Exécution du programme où les travaux sont forme de trois tâches périodiques

## Question 12

En partant de l’hypothèse que les trois tâches démarrent en même temps de manière synchronisée, montrons que les trois tâches s’exécutent toujours correctement avec EDF.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ordre | T2 | T3 | T4 | T2 | T3 | T2 | T4 | T2 | T3 | T2 | T4 | T3 | T2 |
| Exécution | 0.33 | 1 | 2 | 0.33 | 1 | 0.33 | 2 | 0.33 | 1 | 0.33 | 2 | 1 | 0.33 |
| Inférieure | 0 | 0 | 0 | 2 | 3 | 4 | 4 | 6 | 6 | 8 | 8 | 9 | 10 |
| Écoulé | 0.33 | 1.33 | 3.33 | 3.67 | 4.67 | 5 | 7 | 7.33 | 8.33 | 8.67 | 10.67 | 11.67 | 12 |
| Sup | 2 | 3 | 4 | 4 | 6 | 6 | 8 | 8 | 9 | 10 | 12 | 12 | 12 |

Les tâches respectent leurs contraintes de temps et s’exécutent correctement avec EDF.

## Question 13

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ordonnancement** | **Temps d'exécution** | **Temps limite inférieure** | **Temps écoulé** | **Temps limite supérieure** |
| T2 | 0,333 | 0 | 0,333 | 2 |
| T3 | 1 | 0 | 1,333 | 3 |
| T4 | 0,667 | 0 | 2 | 4 |
| T2 | 0,333 | 2 | 2,333 | 4 |
| T4 | 0,667 | 0 | 3 | 4 |
| T3 | 1 | 3 | 4 | 6 |

On a une erreur avec T4.

Priorité : T2, T3, T4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ordonnancement** | **Temps d'exécution** | **Temps limite inférieure** | **Temps écoulé** | **Temps limite supérieure** |
| T4 | 2 | 0 | 2 | 4 |

ERREUR ! T2

Priorité : T4…

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ordonnancement** | **Temps d'exécution** | **Temps limite inférieure** | **Temps écoulé** | **Temps limite supérieure** |
| T3 | 1 | 0 | 1 | 3 |
| T2 | 0,333 | 0 | 1,333 | 2 |
| T4 | 0,667 | 0 | 2 | 4 |
| T2 | 0,333 | 2 | 2,333 | 4 |
| T3 | 1 | 3 | 3,333 | 6 |
| T4 | 1,333 | 0 | 4,666 | 4 |
| On a une erreur | Avec T4 |  |  |  |

Priorité : T3, T2, T4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ordonnancement** | **Temps d'exécution** | **Temps limite inférieure** | **Temps écoulé** | **Temps limite supérieure** |
| T3 | 1 | 0 | 1 | 3 |
| T4 | 2 | 0 | 3 | 2 |
| ERREUR ! T2 ! |  |  |  |  |

Priorité : T3, T4, T2

On voit donc qu’il n’y a pas de solution pour donner un ordre de priorité des tâches. Nous avons obtenu une erreur à chaque fois.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ordonnancement** | **Temps d'exécution** | **Temps limite inférieure** | **Temps écoulé** | **Temps limite supérieure** |
| T2 | 0,22311 | 0 | 0,22311 | 2 |
| T3 | 0,67 | 0 | 0,89311 | 3 |
| T4 | 1,10699 | 0 | 2,0001 | 4 |
| T2 | 0,22311 | 2 | 2,22321 | 4 |
| T4 | 0,23301 | 0 | 2,45622 | 4 |

Avec un ordinateur qui va 1.3 fois plus vite on arrive à donner un ordre des priorités sur les tâches étant donné que l’ensemble des tâches est réalisé en moins de 3 secondes alors que ces dernières devaient se finir en 4 secondes maximums dans l’exemple ci-dessus.

## Question 14