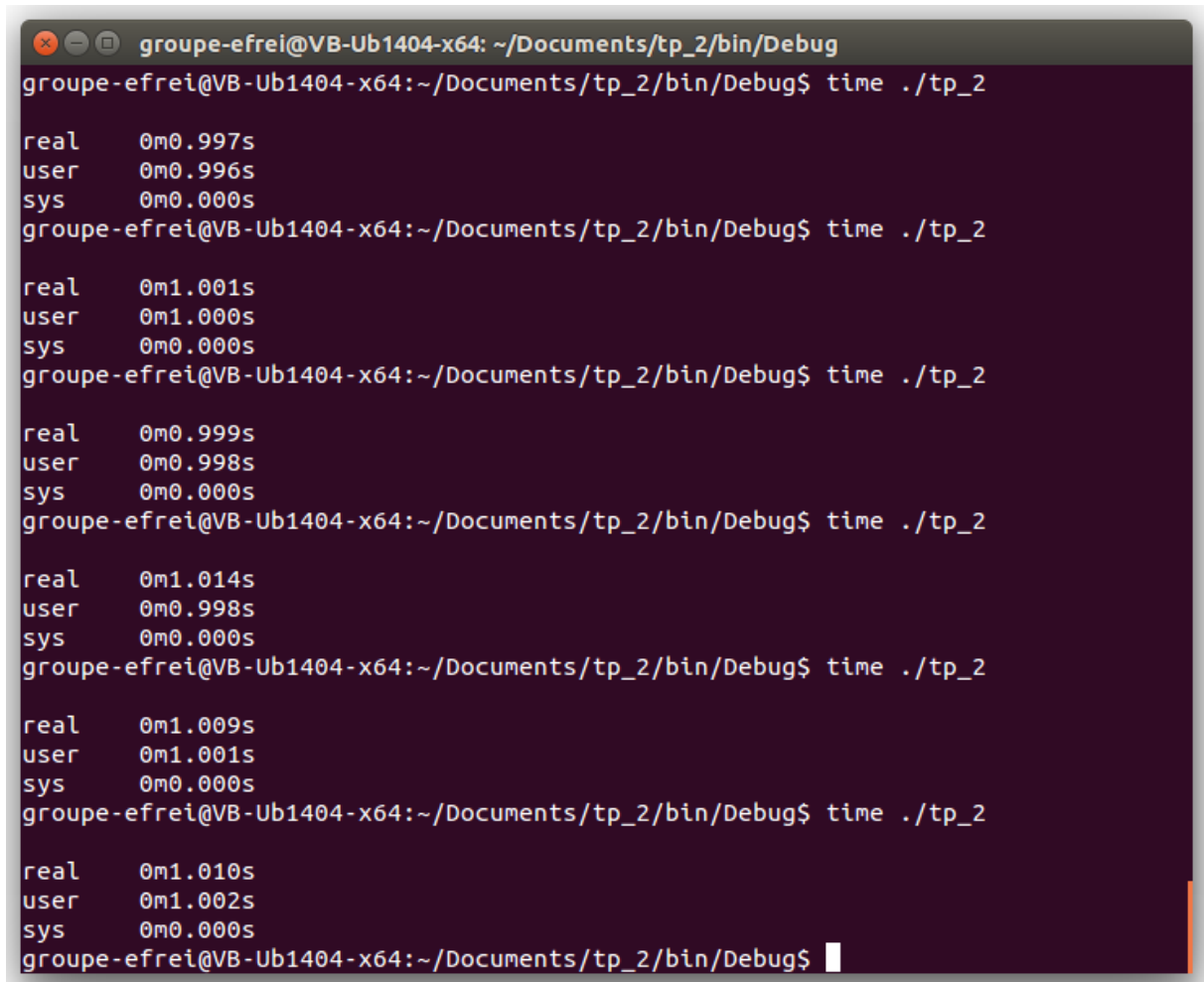


Rapport Real-Time Systems: TP2

Question 1

A terminal window with a dark purple background and white text. The window title is 'groupe-efrei@VB-Ub1404-x64: ~/Documents/tp_2/bin/Debug'. The terminal shows six consecutive executions of the command 'time ./tp_2'. Each execution outputs three lines: 'real', 'user', and 'sys', followed by their respective time values in seconds. The 'real' time values are approximately 0.997s, 1.001s, 0.999s, 1.014s, 1.009s, and 1.010s. The 'user' and 'sys' times are consistently around 0.996s to 1.002s and 0.000s respectively.

```
groupe-efrei@VB-Ub1404-x64: ~/Documents/tp_2/bin/Debug
groupe-efrei@VB-Ub1404-x64:~/Documents/tp_2/bin/Debug$ time ./tp_2

real    0m0.997s
user    0m0.996s
sys     0m0.000s
groupe-efrei@VB-Ub1404-x64:~/Documents/tp_2/bin/Debug$ time ./tp_2

real    0m1.001s
user    0m1.000s
sys     0m0.000s
groupe-efrei@VB-Ub1404-x64:~/Documents/tp_2/bin/Debug$ time ./tp_2

real    0m0.999s
user    0m0.998s
sys     0m0.000s
groupe-efrei@VB-Ub1404-x64:~/Documents/tp_2/bin/Debug$ time ./tp_2

real    0m1.014s
user    0m0.998s
sys     0m0.000s
groupe-efrei@VB-Ub1404-x64:~/Documents/tp_2/bin/Debug$ time ./tp_2

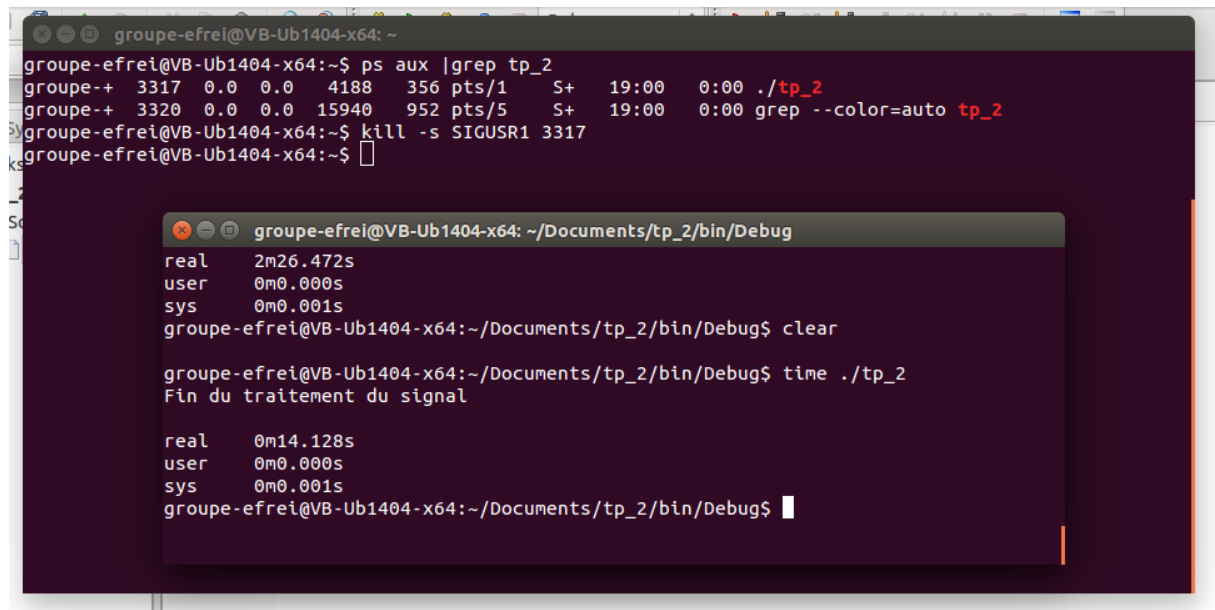
real    0m1.009s
user    0m1.001s
sys     0m0.000s
groupe-efrei@VB-Ub1404-x64:~/Documents/tp_2/bin/Debug$ time ./tp_2

real    0m1.010s
user    0m1.002s
sys     0m0.000s
groupe-efrei@VB-Ub1404-x64:~/Documents/tp_2/bin/Debug$
```

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.

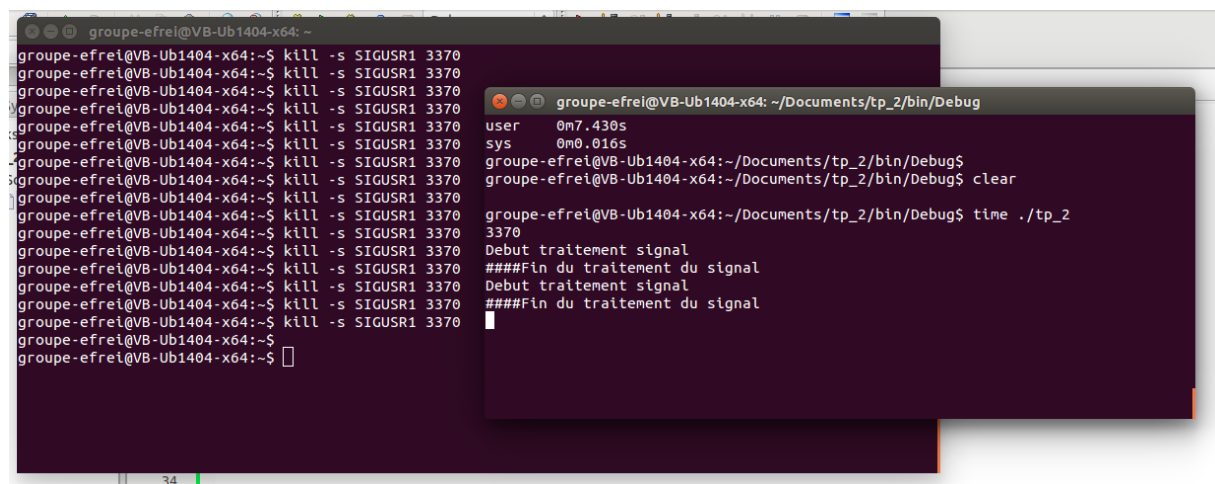
Question 2



```
groupe-efrei@VB-Ub1404-x64: ~  
groupe-efrei@VB-Ub1404-x64:~$ ps aux |grep tp_2  
groupe-+ 3317 0.0 0.0 4188 356 pts/1 S+ 19:00 0:00 ./tp_2  
groupe-+ 3320 0.0 0.0 15940 952 pts/5 S+ 19:00 0:00 grep --color=auto tp_2  
groupe-efrei@VB-Ub1404-x64:~$ kill -s SIGUSR1 3317  
groupe-efrei@VB-Ub1404-x64:~$  
  
groupe-efrei@VB-Ub1404-x64: ~/Documents/tp_2/bin/Debug  
real 2m26.472s  
user 0m0.000s  
sys 0m0.001s  
groupe-efrei@VB-Ub1404-x64:~/Documents/tp_2/bin/Debug$ clear  
  
groupe-efrei@VB-Ub1404-x64:~/Documents/tp_2/bin/Debug$ time ./tp_2  
Fin du traitement du signal  
  
real 0m14.128s  
user 0m0.000s  
sys 0m0.001s  
groupe-efrei@VB-Ub1404-x64:~/Documents/tp_2/bin/Debug$
```

Lancement du programme et envoi d'un signal SIGUSR1 via la commande kill

Question 3



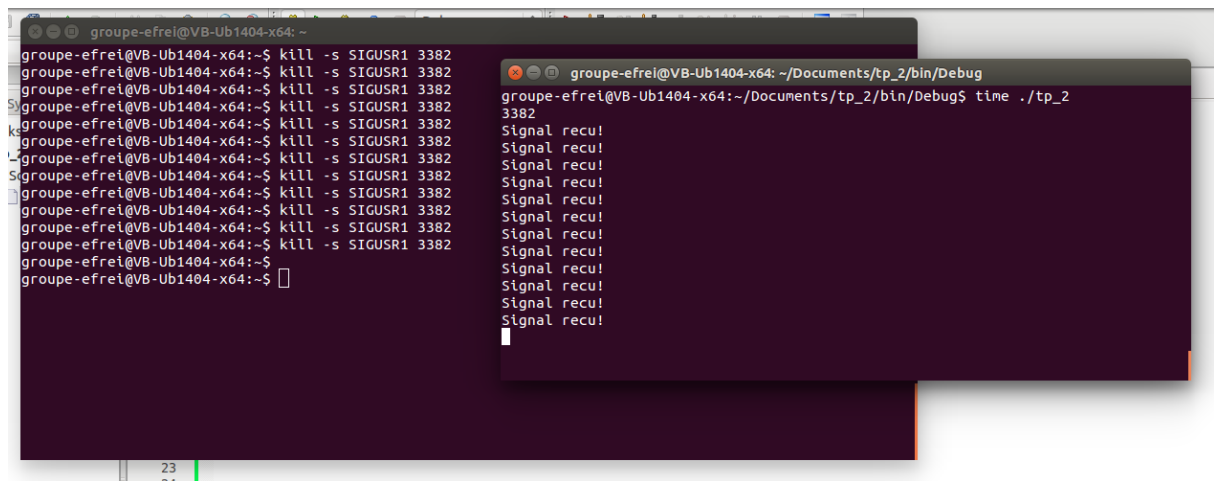
```
groupe-efrei@VB-Ub1404-x64: ~  
groupe-efrei@VB-Ub1404-x64:~$ kill -s SIGUSR1 3370  
groupe-efrei@VB-Ub1404-x64:~$ kill -s SIGUSR1 3370  
groupe-efrei@VB-Ub1404-x64:~$ kill -s SIGUSR1 3370  
groupe-efrei@VB-Ub1404-x64:~$ kill -s SIGUSR1 3370  
groupe-efrei@VB-Ub1404-x64:~$ kill -s SIGUSR1 3370  
groupe-efrei@VB-Ub1404-x64:~$ kill -s SIGUSR1 3370  
groupe-efrei@VB-Ub1404-x64:~$ kill -s SIGUSR1 3370  
groupe-efrei@VB-Ub1404-x64:~$ kill -s SIGUSR1 3370  
groupe-efrei@VB-Ub1404-x64:~$ kill -s SIGUSR1 3370  
groupe-efrei@VB-Ub1404-x64:~$ kill -s SIGUSR1 3370  
groupe-efrei@VB-Ub1404-x64:~$ kill -s SIGUSR1 3370  
groupe-efrei@VB-Ub1404-x64:~$ kill -s SIGUSR1 3370  
groupe-efrei@VB-Ub1404-x64:~$ kill -s SIGUSR1 3370  
groupe-efrei@VB-Ub1404-x64:~$ kill -s SIGUSR1 3370  
groupe-efrei@VB-Ub1404-x64:~$ kill -s SIGUSR1 3370  
groupe-efrei@VB-Ub1404-x64:~$ kill -s SIGUSR1 3370  
groupe-efrei@VB-Ub1404-x64:~$ kill -s SIGUSR1 3370  
groupe-efrei@VB-Ub1404-x64:~$  
  
groupe-efrei@VB-Ub1404-x64: ~/Documents/tp_2/bin/Debug  
user 0m7.430s  
sys 0m0.016s  
groupe-efrei@VB-Ub1404-x64:~/Documents/tp_2/bin/Debug$  
groupe-efrei@VB-Ub1404-x64:~/Documents/tp_2/bin/Debug$ clear  
  
groupe-efrei@VB-Ub1404-x64:~/Documents/tp_2/bin/Debug$ time ./tp_2  
3370  
Debut traitement signal  
####Fin du traitement du signal  
Debut traitement signal  
####Fin du traitement du signal  
█
```

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 reçues 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



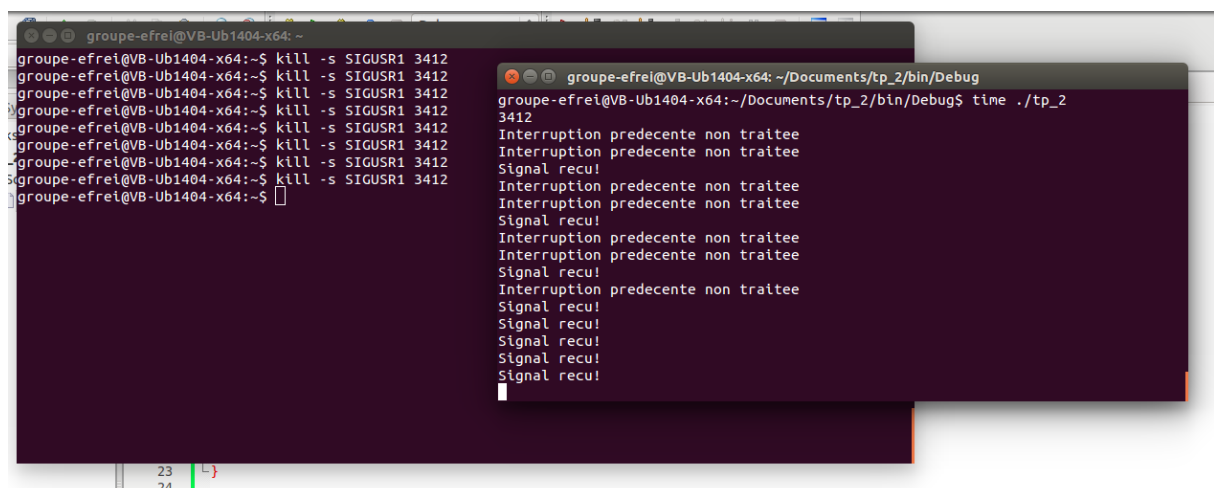
The screenshot shows two terminal windows. The left window is a terminal with the prompt 'groupe-efrei@VB-Ub1404-x64: ~' and contains 12 lines of the command 'kill -s SIGUSR1 3382'. The right window is a terminal with the prompt 'groupe-efrei@VB-Ub1404-x64: ~/Documents/tp_2/bin/Debug' and contains the command 'time ./tp_2 3382' followed by 12 lines of 'Signal recu!'.

Envoi 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 a 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 où le gestionnaire d'interruption a déjà intercepté un signal alors il ne pourra pas le traiter.

Question 5



The screenshot shows two terminal windows. The left window is a terminal with the prompt 'groupe-efrei@VB-Ub1404-x64: ~' and contains 8 lines of the command 'kill -s SIGUSR1 3412'. The right window is a terminal with the prompt 'groupe-efrei@VB-Ub1404-x64: ~/Documents/tp_2/bin/Debug' and contains the command 'time ./tp_2 3412' followed by 8 lines of 'Signal recu!' and 8 lines of 'Interruption precedente non traitee'.

Envoi 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

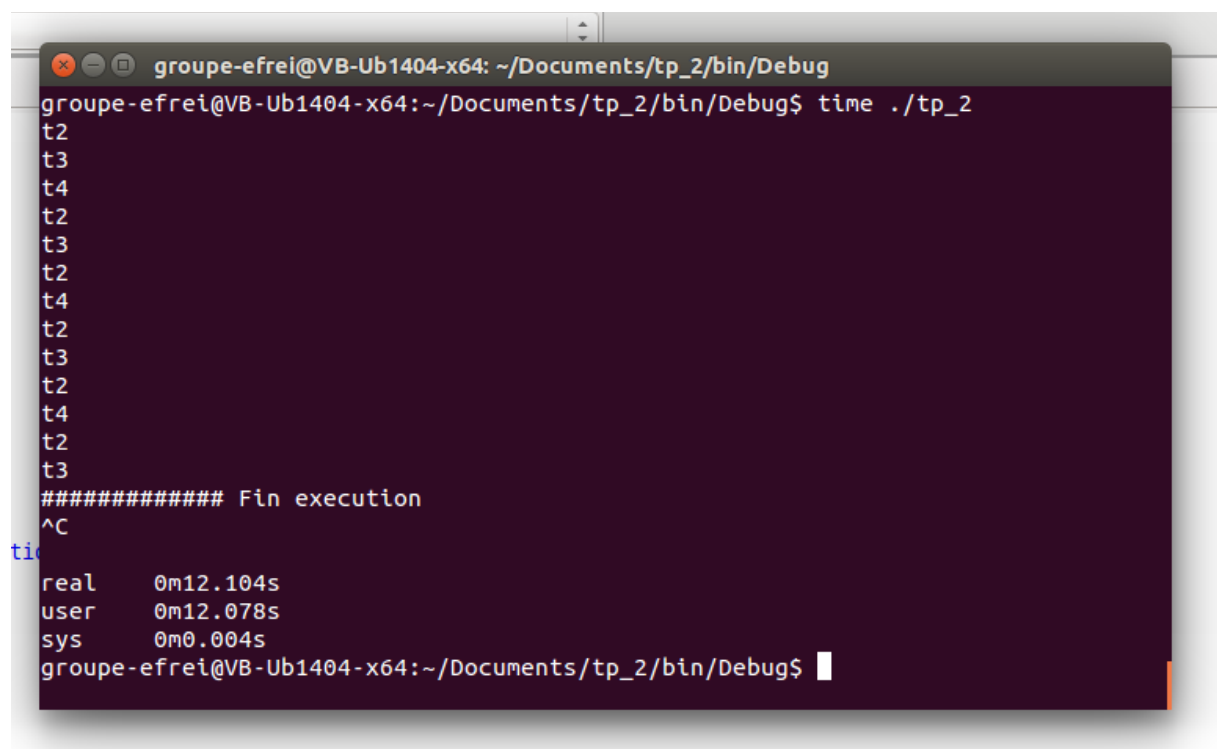
Fabien Beaujean
Adrien Poupa

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



```
groupe-efrei@VB-Ub1404-x64: ~/Documents/tp_2/bin/Debug
groupe-efrei@VB-Ub1404-x64:~/Documents/tp_2/bin/Debug$ time ./tp_2
t2
t3
t4
t2
t3
t2
t4
t2
t3
t2
t4
t2
t3
##### Fin execution
^C
real    0m12.104s
user    0m12.078s
sys     0m0.004s
groupe-efrei@VB-Ub1404-x64:~/Documents/tp_2/bin/Debug$
```

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 court) 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.

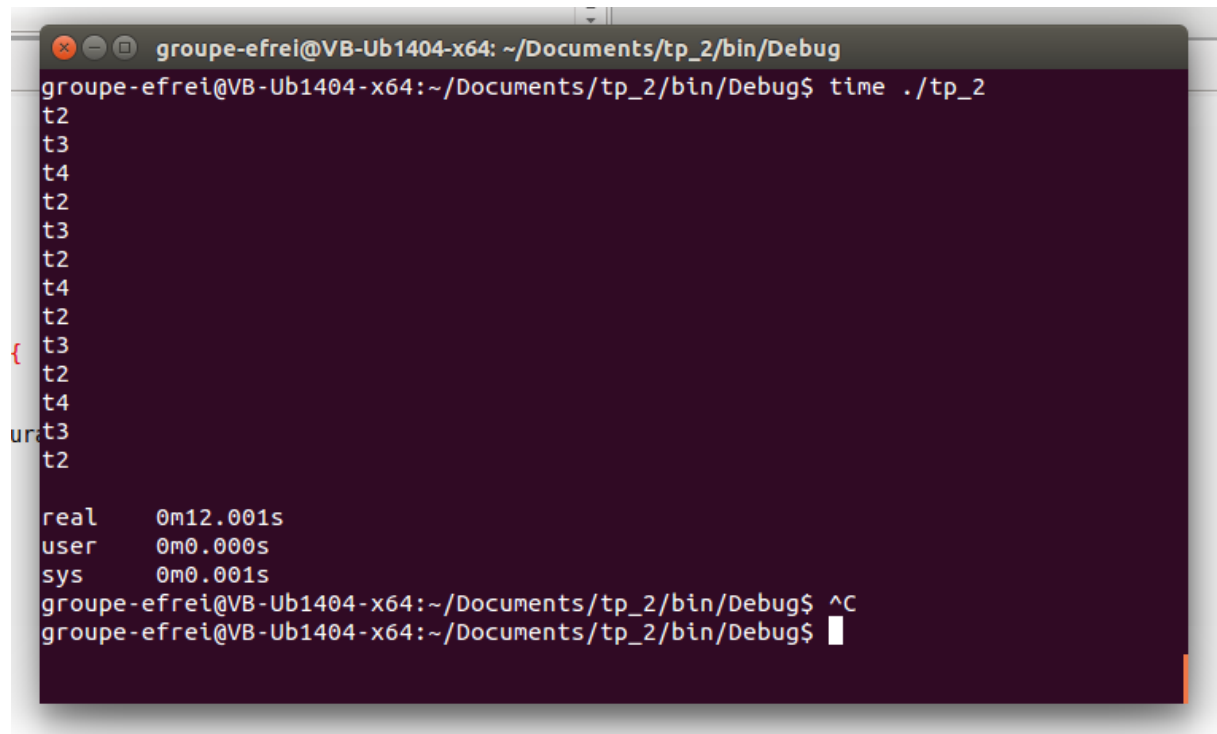
Pour résoudre le problème, il faut insérer entre chaque travail des pauses dont le temps devra être calculé indépendamment de la vitesse du processeur et donc empêcher l'exécution d'une tâche avant que ce temps d'attente ait été exécuté

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



```
groupe-efrei@VB-Ub1404-x64: ~/Documents/tp_2/bin/Debug
groupe-efrei@VB-Ub1404-x64:~/Documents/tp_2/bin/Debug$ time ./tp_2
t2
t3
t4
t2
t3
t2
t4
t2
t3
t2
t4
t3
t2
t4
t3
t2
real    0m12.001s
user    0m0.000s
sys     0m0.001s
groupe-efrei@VB-Ub1404-x64:~/Documents/tp_2/bin/Debug$ ^C
groupe-efrei@VB-Ub1404-x64:~/Documents/tp_2/bin/Debug$
```

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

Fabien Beaujean
Adrien Poupa

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.