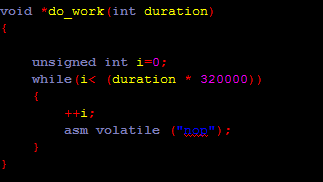
|  |
| --- |
| TP2 – Système temps réels |
| FABRE Adrien & CHEKROUN Thomas |
|  |

03/04/2015

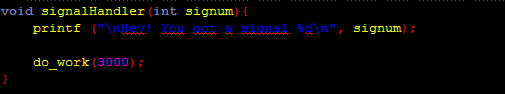
Question 1

La function “do-work” va nous permettre de simuler une action d’une duée « duration » exprimé en milliseconde.

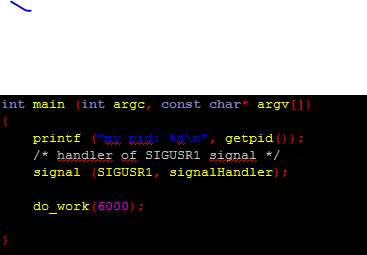


Question 2

Pour faire que notre programme affiche un message dès qu’il reçoit un message de type « SIGUSR1 » il suffit de créer une fonction « signalHandler » qui va effectuer une tâche, ici afficher un message puis simuler un travailler d’une durée de 3000ms.



Pour que cette fonction soit lancée dès la réception d’un signal « SIGUSR1 » il faut rajouter un appel à cette fonction dans le main.



L’appel à la méthode « do\_work » permet que le programme continue de tourner, car si le programme ne reçoit pas de message il s’arrête alors que ici le programme va avoir un temps d’attente.

Question 3

On rajoute une exécution de la fonction do\_work() dans le signal handler pour simuler un traitement long. En envoyant une rafale de signaux on remarque que tous les signaux envoyés ne sont pas interceptés. Cela vient du fait que l'OS utilise un bit à 0 ou à 1 pour indiquer qu'un signal a été reçu. Ainsi si le traitement du signal est long le bit n'a pas le temps d'être remis à 0 et certains signaux ne sont donc pas "vus". La condition serait que les signaux soit envoyé avec un intervalle de temps assez grand pour que le « do\_work » soit terminé. Dans notre exemple on pourrait envoyer des signaux avec un intervalle de temps de 1500ms et le programme arrivera à capter tous les signaux.

Question 4

Pour cette question on peut créer une variable globale qui nous servira de compteur, à chaque reception de signal on incrémente ce compteur puis ensuite dans le main on fait une boucle qui le décrémente en faisant le « do\_work »

// A implementer

Mais le problème ne sera pas totalement résolu car il faudra que les signaux n’arrivent pas avec un intervalle de temps inférieur à l’incrémentation d’un compteur

Question 5

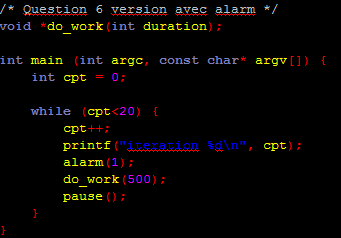
No idea :D Have Fun

Question 6

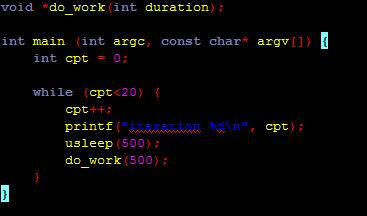
La fonction usleep(500ms) va nous permettre de mettre en pause le programme pendant 0,5 sec donc si on lui rajoute un do\_work(500) cela va nous donner une tâche périodique de 1s.

La différence avec la fonction alarm() c’est que alarm() va envoyer un signal de type « SIGALARM » au bout d’un certain temps. Mais pour capter le signal il faudra mettre une « pause() » qui va endormir le thread jusqu’à la réception d’un signal. Entre l'appel d’alarm () et pause() on fait notre do\_work. Par défaut SIGALRM tue le processus il faudra donc rajouter un signal handler pour changer ce comportement.

Le problème de ces méthodes est que l’on fait appel à des fonctions externes du coup on risque un décalage dans le temps.



Version avec usleep :



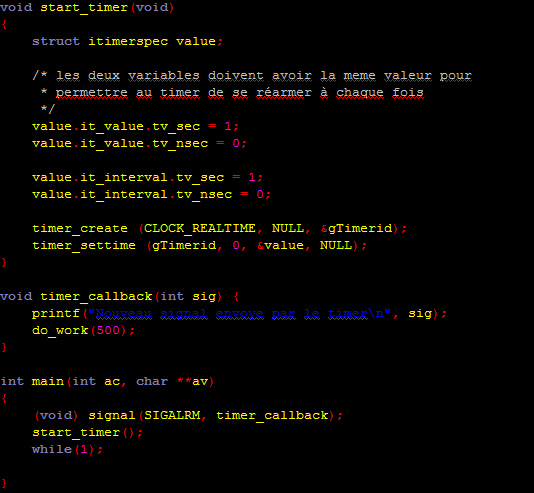
Question 7

La fonction « startTimer » nous permet de créer notre timer, on crée la valeur de notre timer (de type struct itimerspec ) qui prend 2 arguments.

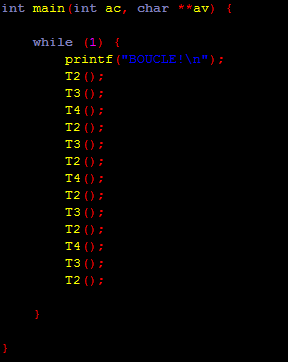
Si on veut un timer de façon infini il faut mettre les deux variables à la même valeur.

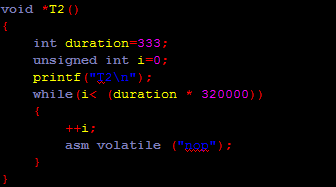
* Value = 0 le timer ne se lancera jamais,
* Si l’intervalle est a 0 il ne se lancera qu’une seule fois.

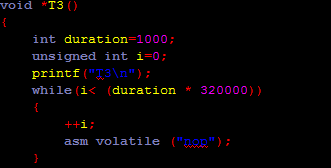
On appel ensuite timer create, et settime pour lui attribuer les attributs définis. Toute les secondes le timer va lancer un SIGALARM qui va appeler la methode timer\_callBack() et va faire notre do\_work.

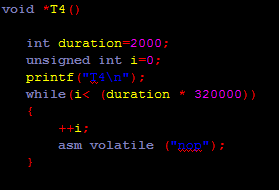


Question 8









Question 9