[TD-1] Mesure de temps et échantillonnage en temps

a) Gestion simplifiée du temps Posix

Cet exercice est destiné à produire des fonctions et opérateurs permettant une utilisation simplifiée de la structure `timespec` représentant la mesure des temps dans l'API Posix. Pour accéder à cette structure, vous devez inclure le header `time.h`. Dans toutes les fonctions suivamtes, les variables de type `timespec` résultant d'un calcul doivent être normalisées avec les contraintes suivantes:

- le champ `tv_sec` (de type `time_t`) peut prendre des valeurs négatives
- le champ `tv_nsec` (de type `long`) doit être positif ou nul et toujours inférieur à `1000000000` (10^9) et ce, même pour un temps globalement négatif.

> Déclarez et implémentez les fonctions suivantes permettant de convertir des `millisecondes` en `timespec` et vice-versa:

```
double timespec_to_ms(const timespec& time_ts);
timespec timespec_from_ms(double time_ms)
```

> Déclarez et implémentez les fonctions suivantes permettant respectivement d'obtenir le temps courant et l'opposé d'un temps:

```
timespec timespec_now();
timespec timespec_negate(const timespec& time_ts);
```

> Déclarez et implémentez les fonctions suivantes permettant respectivement d'ajouter et de soustraire deux `timespec`:

```
timespec timespec_add(const timespec& time1_ts, const timespec& time2_ts);
timespec timespec_subtract(const timespec& time1_ts, const timespec& time2_ts);
```

> En utilisant la fonction Posix `nanosleep()`, déclarez et implémentez la fonction suivante permettant d'endormir la tâche appelante durant le temps spécifié; attention, il faut gérer l'éventualité où l'appel de `nanosleep()` soit interrompu (voir `man nanosleep`):

```
timespec timespec_wait(const timespec& delay_ts);
```

> En utilisant les fonctions ci-dessous, déclarez et implémentez les opérateurs ci-dessous afin de pouvoir utiliser les variables `timespec` comme des scalaires:

```
timespec  operator- (const timespec& time_ts);
timespec  operator+ (const timespec& time1_ts, const timespec& time2_ts);
timespec  operator- (const timespec& time1_ts, const timespec& time2_ts);
timespec& operator+= (timespec& time_ts, const timespec& delay_ts);
timespec& operator-= (timespec& time_ts, const timespec& delay_ts);
bool operator== (const timespec& time1_ts, const timespec& time2_ts);
bool operator!= (const timespec& time1_ts, const timespec& time2_ts);
bool operator< (const timespec& time1_ts, const timespec& time2_ts);
bool operator> (const timespec& time1_ts, const timespec& time2_ts);
```

> Écrivez un programme `main()` permettant de tester toutes ces fonctions.

b) Timers avec callback

Pour cette question, reportez-vous aux explications du cours sur les timers Posix avec callback.

> Implémentez un timer Posix périodique de fréquence 2 Hz imprimant un message avec la valeur d'un compteur régulièrement incrémenté.
> Le programme doit s'arrêter après 15 incrémentations.

c) Fonction simple consommant du CPU

On considère la fonction de signature : `void incr(unsigned int nLoops, double* pCounter)`. Cette fonction doit effectuer une boucle incrémentant de `1.0` la valeur du compteur pointée par `pCounter` elle doit effectuer cette boucle `nLoops` fois.

> Écrivez le code de cette fonction.

La signature standard du point d'entrée d'un programme est : `int main(int argc, char* argv[])`. Le paramètre `argc` indique le nombre de chaines de caractères de la ligne de commande ayant lancé l'exécution du programme ; le paramètre `argv` est le tableau de l'ensemble de ces chaines dans l'ordre où ils ont été tapés sur la ligne de commande, `argv[0]` étant le nom du programme, `argv[1]` son 1er paramètre, `argv[2]` son 2e paramètre, etc. Ici, la fonction `main` doit :

– déclarer une variable `nLoops` et l'initialiser avec la valeur numérique décimale de `argv[1]` ;
– déclarer un compteur `counter` de type `double` et l'initialiser à `0.0` ;
– appeler la fonction `incr` sur ces deux variables ;
– imprimer à l'écran la valeur finale de `counter`.

> Écrivez, compilez et exécutez le programme ainsi défini. Notez la valeur finale du compteur `counter`.

Renseignez-vous sur la fonction Posix `clock_gettime` exposée dans le cours.

> En utilisant `clock_gettime`, affichez à la fin du programme le temps d'exécution de la fonction `incr` ; affichez ce temps en secondes sous la forme d'un nombre à virgule.

d) Mesure du temps d'exécution d'une fonction

On reprend la fonction `incr` définie **ci-dessus** et on la modifie en lui rajoutant un 3e paramètre et une valeur de retour :
`unsigned incr(unsigned int nLoops, double* pCounter, bool* pStop)`.

Comme avant, cette fonction effectue dans sa boucle principale une incrémentation de `1.0` de la valeur du compteur passé en paramètre ; la différence avec l'implémentation précédente est que la condition d'arrêt peut être atteinte **avant** d'avoir effectué le nombre maximal de boucles (`nLoops`), dans le cas où le paramètre pointé par `pStop` passe à la valeur `true`. Sa valeur de retour est le nombre de boucles effectuées (≤ `nLoops`).

> Modifiez l'implémentation de la fonction `incr` conformément aux indications ci-dessus.

Ce paramètre `pStop` est destiné à effectuer des mesures du nombre de boucles effectuées par la fonction `incr` durant des temps d'exécution bien déterminés :

– on déclare dans le `main` une variable booléenne `stop` initialisée à `false` et dont on passera l'adresse en paramètre de `incr` en tant que `pStop` ;
– avant d'appeler `incr`, on initialise `nLoops` à sa valeur maximale possible (utilisez la constante `UINT_MAX` définie dans le header standard `<climits>`) ;
– on lance un timer sur un temps bien défini (par exemple 1 seconde) avec un callback dont la fonction est de faire passer la valeur de `stop` à `true` : la boucle de `incr` s'arrêtera alors et renverra le nombre de boucles effectuées durant ce temps.

> Que faut-il modifier dans la déclaration de `pStop` ?

Soit $l(t)$ le nombre de boucles effectuées par la fonction `incr` durant l'intervalle de temps t ; on suppose que cette fonction est affine : $l(t)=a\times t+b$. Soit `iLoop` la variable comptant la boucle dans la fonction `incr` (dans l'expression `for(iLoop=0 ; iLoop < nLoops ; ++iLoop)`)

> Implémentez une fonction `calib` établissant les valeurs de a et b en mesurant la valeur de `iLoop` pour 4 secondes et pour 6 secondes.
> Vérifiez par programme (en utilisant la fonction `clock_gettime` vue **ci-dessus** que votre calibration est correcte.

e) Amélioration des mesures

> Imaginez des solutions pour améliorer la précision la fonction $l(t)$:
> – en effectuant plus de mesures
> – en vous assurant que l'exécution de la fonction `calib` ne puisse pas être perturbée par d'autres tâches.