C - tutorat - 2

Arthur BOYREAU

September 22, 2026

Contents

1 Sujet **1**
1.1 Horloge 1
1.2 Les tâches 1
1.2.1 Afficheur 1
1.2.2 Nettoyeur 2

2 Vers une meilleure organisation du code **2**
2.1 Proposition d'architecture 2

3 Une première interface **3**

1 Sujet

On veut implémenter un planificateur de tache. On part d'une horloge et de quelques tâches:

- afficher le temps actuel (en seconde)
- nettoyer l'affichage (clear)

1.1 Horloge

Le coeur de notre application est une horloge, qui vérifie chaque seconde si des tâches sont à exécuter ou non.

Elle peut être implémentée de manière assez simple à l'aide des instructions suivantes :

1. while(1) (on veut boucler pour toujours, sans jamais s'arrêter).
2. int sleep(int delay); : on vérifiera chaque seconde si des tâches sont en attentes, et entre 2 vérifications, on attendra !
3. int time(int *ptr); : pour récupérer le temps actuel (en seconde). Le pointeur ptr peut être NULL.

Écrivez une horloge simple dans le main de votre programme.

1.2 Les tâches

1.2.1 Afficheur

En utilisant la fonction time et la fonction printf (*mantime* et *manprintf* sont à votre disposition), affichez le temps actuel (en seconde, pas besoin d'aller chercher trop loin avec le format).

Cette tâche doit être planifiée chaque seconde.

Ajoutez cette tache à votre code.

1.2.2 Nettoyeur

En utilisant la fonction *system* (*mansystem*), exécutez la commande shell "clear" pour effacer l'écran.

Cette tâche doit être planifiée toutes les 3 secondes.

Ajoutez cette tache à votre code.

Est-il facile d'ajouter des tâches ?

Le code vous parait-il propre ?

2 Vers une meilleure organisation du code

On arrive sur un point clé et fondamental : le refactor (ou la réorganisation du code).

C'est le moment de prendre un peu de recul, de relire son code et de le réorganiser pour se faciliter la vie.

Les principaux enjeux de notre code seront :

- Planifier facilement des tâches
- Ajouter des tâches

On peut suivre le principe suivant : à chaque fonction sa responsabilité (aka, une tache = une fonction).

Que faudrait-il faire pour faciliter l'ajout et la suppression de tâches ?

2.1 Proposition d'architecture

Mon projet est actuellement dans cet état :

```
$> tree
.
|-- bonus.tex
|-- compile_commands.json
|-- include
|   |-- horloge.h   # Déclare la fonction permettant d'exécuter les tâches planifiées.
```

```
|   |-- afficheur.h # Déclare la fonction d'affichage.
|   |-- nettoyeur.h # Déclare la fonction de nettoyage.
|-- main
|-- Makefile
|-- README.md
|-- src
    |-- main.c
    |-- horloge.c   # Le code du planificateur de tache. Il appelle les tâches directeme
    |-- afficheur.c # Le code de la tache affichant l'heure (une seule fonction).
    |-- nettoyeur.c # Le code de la tache nettoyant l'écran (une seule fonction).
3 directories, 12 files
```

Regard critique sur moi-même : un sous-dossier taches pourrait être de mise...
En l'état, on peut facilement ajouter de nouvelles tâches au code d'un pdv architecture...

Mais à la moindre erreur de planification dans une tâche, toute l'horloge peut partir en vrille et toutes les taches peuvent se décaler, c'est pas terrible.
→ Il faudrait ne pas modifier directement la fonction horloge !

Comment modifier horloge.c et horloge.h pour pouvoir ajouter des tâches à la volée sans avoir à remettre le code de l'horloge en question à chaque nouvelle tache ? Indice : l'alpha et l'omega de l'objet...

3 Une première ...

Jusqu'à maintenant, nos tâches ne prennent pas de paramètres, et ne retournent rien. On a donc des fonctions de type

```
void␣*tache(void);
```

L'idée serait ...