# Travaux Pratiques & Travaux Dirigés
# Temps-Réel et Parallélisme

## Partie 1

## *Objectifs*

Les objectifs de ce TP sont multiples :

- Concevoir une application multitâche (non temps réel) avec une approche dirigée par les événements.
- Manipuler les mécanismes d'un système d'exploitation dit « asynchrone » pour implémenter cette application.
- Concevoir cette même application avec une approche dirigée par le temps
- Utiliser les mécanismes d'une chaine de production dirigée par le temps et orientée sûreté de fonctionnement pour implémenter cette application
- Synthétiser les avantages/inconvénients de chacune des approches.
- Aborder les problèmes d'interblocage et de famine dans les approches par événements

*Note : Nous ne cherchons pas à faire dans ce TP une application temps réel. Ainsi, pour l'approche dirigée par les événements, nous allons utiliser un système d'exploitation classique Unix et les apis normatives POSIX. Les apis que vous allez manipuler, les phénomènes que vous allez observer et la conception que vous allez mettre en œuvre dans ce TP seraient reproductibles sur un système d'exploitation dit « Temps Réel ».*

## *Sources*

Le code source pour ce tp est présent à l'adresse suivante :

https://github.com/fthomasfr/multitasking_training_practical_work

Pour le récupérer vous pouvez le télécharger directement ou utiliser git :

```
git clone https://github.com/fthomasfr/multitasking_training_practical_work.git
```

L'ensemble des informations et codes d'exemples concernant la programmation multitâche avec les apis POSIX sont disponibles en ligne : http://pubs.opengroup.org/onlinepubs/9699919799/

# *TP 1ere Partie*

Nous devons réaliser un logiciel sûr de fonctionnement satisfaisant la spécification et l'architecture suivante :

- Notre logiciel possède quatre entrées numériques asynchrones, une sortie numérique et une sortie de diagnostic. Une entrée est caractérisée par un tableau de 256 entiers et un checksum. La sortie numérique est caractérisée par un tableau de 256 entiers. La sortie diagnostic est une chaine de caractères à destination du terminal.
- **Exigence 1 :** Le logiciel doit acquérir quatre entrées et produire en sortie le cumul des entrées. Le cumul correspond à la production d'un tableau de 256 entiers dont le $i^{eme}$ élément de ce tableau est la somme des $i^{emes}$ éléments des tableaux d'entrée.
- **Exigence 2 :** Le logiciel doit acquérir toutes les données d'entrées
- **Exigence 3 :** Le logiciel doit garantir que les données d'entrée sont correctement formées avant de les sommer.
- **Exigence 4 :** Le logiciel doit sommer et produire une sortie au plus vite, c'est-à-dire dès qu'une entrée est présente sans attendre une nouvelle valeur sur chacune des entrées.
- **Exigence 5 :** Le logiciel doit produire sur la sortie diagnostic pour chaque opération de cumul, combien d'entrées ont été acquises, combien ont été sommées et combien restent à sommer.

## Conception en utilisant une approche dirigée par les événements

Plusieurs conceptions peuvent exister. Nous proposons de guider votre conception en suivant les choix de conception suivants:

- Utilisation de tâches où la mémoire est partagée entre les tâches
- Partage d'un tableau de messages. Un message est une entrée acquise
- Utilisation d'événements sans transmission de données pour synchroniser les tâches
- Utilisation d'un checksum pour satisfaire l'exigence 3
- L'utilisation d'une librairie de messages. La fonction MessageFill permet de simuler la production d'une entrée. La fonction MessageDisplay permet d'afficher le message.

1. Etudiez la librairie de messages fournis (fichier msg.h et msg.c) pour en comprendre les structures de données et les apis.
    a. Pourquoi certaines variables sont-elles considérées comme des variables C volatiles ?
2. Ecrivez un chapitre d'architecture logiciel (Software Architecture Document) allouant les exigences précédentes.
3. Ecrivez un chapitre d'architecture logiciel détaillée (Detailled design) satisfaisant l'ensemble des exigences précédentes et utilisant les directives ci-dessus. Cette conception devra utiliser des diagrammes de composants pour décrire l'architecture structurelle et des diagrammes de séquences pour expliquer et démontrer la causalité des traitements (architecture comportementale).
4. Nous avons orienté la conception par l'utilisation des tâches. Nous aurions pu utiliser des processus. Citez les caractéristiques intéressantes des processus pour une conception orientée « sûreté de fonctionnement » ? Citez également celles qui ne sont pas satisfaites ? Auriez-vous pu utiliser les mêmes choix de conception que ceux-ci-dessus ?

## Implémentation dirigée par les événements

5. Implémentez votre conception (Implémentation + Exécution)

6. En quoi votre conception et votre implémentation ne satisfait pas les exigences de sûreté de fonctionnement vues en cours ?

## Pour aller plus loin en conception et implémentation dirigées par les événements

7. Comment auriez-vous pu protéger de manière efficace, sûr, sans mutex et sans sémaphores les compteurs permettant d'indiquer pour chaque consommation, combien de messages ont été produits, combien ont été consommés et combien reste à consommer ?
    a. Pour cela réfléchissez à ce que le compilateur et le processeur peuvent nativement proposer
        i. Le compilateur GCC donne accès ainsi à une série de fonction dans https://gcc.gnu.org/onlinedocs/gcc-4.1.0/gcc/Atomic-Builtins.html#Atomic-Builtins Utiliser dans votre programme certaines de ces APIs.