# Ordonnancement de threads

***Mots-clés** : ordonnancement, exclusion mutuelle, multithread, synchronisation*

## Spécifications du problème

On souhaite gérer l'exécution de tâches (threads) à l'intérieur d'une même application. Les tâches concernées doivent effectuer un traitement en exclusion mutuelle et ont une priorité fixe déterminée au lancement.

Ce fonctionnement sera mis en œuvre par des fonctions et/ou macros appelées par l'application.

### Les tâches

Les tâches concernées ont une des 2 structures suivantes :

```
DEBUT_TACHE
while (condition_de_fin_non_remplie) {
        LOCK
        // traitement à effectuer en exclusion mutuelle
        …
        UNLOCK
}
FIN_TACHE
```

ou

```
DEBUT_TACHE
LOCK
// traitement à effectuer en exclusion mutuelle
…
UNLOCK
FIN_TACHE
```

### L'ordonnancement

L'ordonnancement des tâches est à priorités fixes. Une tâche se voit attribuer une priorité lors de son lancement. Les niveaux de priorité sont des entiers de valeur comprise entre 1 et 5.

Pour un même niveau de priorité, l'exécution est en FIFO. La tâche la plus prioritaire est la plus ancienne dans le niveau de priorité le plus élevé.

Une tâche doit pouvoir céder la main à une autre tâche plus prioritaire dès qu'elle n'est plus dans sa section critique. Dans les faits, l'ordre de priorité ne pourra être réévalué que sur les actions DEBUT_TACHE, LOCK, UNLOCK ou FIN_TACHE.

Un time-out peut être prévu pour éviter un éventuel interblocage (cas d'une tâche ne libérant pas la section critique au bout d'un laps de temps donné). Il doit être paramétrable (valeur ou pas de time-out). Dans ce cas, la tâche « coupable » doit être terminée.

**Implémentation**

L'implémentation est donc celle d'une API (bibliothèque de fonctions) offrant les fonctionnalités suivantes :

- Initialisation (appelée au lancement de l'application pour initialiser le mécanisme),
- Lancement d'une tâche,
- DEBUT_TACHE,
- FIN_TACHE,
- LOCK,
- UNLOCK.

Cette API pourra être utilisée dans toute application dont les tâches s'exécutent en exclusion mutuelle et nécessitant la gestion de priorité définie.

**Environnement de développement et d'exécution**

L'API sera développée en C, devra fonctionner sous Linux. Les tâches seront mises en œuvre par des threads Posix.

## Travail demandé

Ce travail doit s'inscrire dans une démarche projet. Comme tout projet, il doit comporter une phase d'**analyse** (qu'est ce que fait l'application?), de **conception** (comment va-t-on traiter techniquement le problème?), de **réalisation** (codage) et de **tests** (permettant de vérifier que l'application fonctionne dans le cas nominal, les cas aux limites et les cas d'erreur).

### 1. Document à produire

Un rapport écrit (forme électronique ou manuscrite acceptées) doit rendre compte des phases du projet précédemment citées.

Son plan idéal est donc le suivant:

- **analyse du problème** : Vous disposez d'un cahier des charges (sujet du BE). Vous le complèterez  En le relisant, il doit être possible de savoir ce que fait l'application dans TOUS les cas d'utilisation.
- **étude technique** (= document de conception) : doit détailler l'architecture de l'application, décrire les choix techniques effectués en les justifiant et en discutant leurs limites, et comporter l'algorithme. N'oubliez pas de traiter en particulier de la gestion des erreurs et des phases de lancement et de terminaison de l'application.
  Cette étude technique doit pouvoir être fournie à un programmeur et lui permettre de coder l'application sans information complémentaire.

  Le plan idéal de cette étude est le suivant :

  Un schéma des différentes tâches et de leurs interactions (communication et synchronisation)

  L'explication des moyens de communication et de synchronisation utilisés (choix et raisons de ce choix)

  L'algorithme de chaque tâche
- **dossier de tests** : celui-ci doit être écrit avant de commencer le code. Il est aussi fourni au programmeur.
- **résultats des tests** : votre code fonctionne-t-il? Y a t-il des problèmes non résolus? D'où proviennent t-ils?
- **bilan du projet** : où en est le projet à la date limite ? Est-il terminé ? Sinon, quels problèmes avez-vous rencontrés, que reste t-il à faire et combien de temps pensez-vous qu'il vous a manqué pour le finaliser ? Cette partie est fondamentale dans le sens où elle montre votre maîtrise de son déroulement.

Les trois premiers items sont à rédiger **AVANT** de coder, même si d'éventuels problèmes techniques rencontrés lors du codage peuvent induire des modifications dans l'étude technique.

### 2. Code de l'application

Le code C de l'application doit être remis. Il doit être lisible et commenté. Il doit fonctionner correctement sous Linux.

## 3. Méthode

- Travail en binôme : vous rendrez un rapport et un code par groupe.
- Encadrement : Bien que ce BE soit à réaliser en autonomie, n'hésitez pas à demander de l'aide à vos enseignants.
  Vous disposez aussi sur e-campus, dans la rubrique « Programmation concurrente : application au système Unix » d'un forum. Posez-y vos questions, j'essaierai d'y répondre au plus vite.
-

## 4. Délais

Le rapport, le code et les résultats des tests doivent être rendus au plus tard à la date qui vous sera précisée en séance de TD**.** Aucun travail rendu hors délai ne sera accepté.