# AI01
# TP3

REDOUIN INNECCO Gabriel
GAMONDELE Maxime
20 novembre 2025

**Ce document synthétise le travail et la réflexion sur le TD de XXX**
**...**

# Table des matières

# 1 Modifications du schéma

Nous avons opéré plusieurs modifications au schéma fourni dans le sujet :

- Introduction d'une structure `FIFO` englobant les `processus`.
- Toutes les fonctions ont été renommées pour améliorer notre compréhension.

## 1.1 Fonctions Processus

### 1.1.1 Init et Free

Les fonctions suivantes :

```
t_processus* creer_processus(int pid, int arrivee, int duree);
void free_processus(t_processus* p);
```

deviennent :

```
t_processus *processus_init(int pid, int arrivee, int duree);
void processus_free(t_processus **process);
void processus_free_recursive(t_processus *process);
```

La fonction `processus_free` reçoit désormais un `t_processus **` afin de libérer la mémoire puis de mettre le pointeur à `NULL`. Une fonction `processus_free_recursive` a été ajoutée pour libérer toute une chaîne de processus.

### 1.1.2 Load

La fonction :

```
t_processus* charger_processus(char* nom_fichier, int* nb_processus);
```

devient :

```
t_processus *processus_load(char *nom_fichier, int *processes_loaded);
```

Elle lit des lignes jusqu'à la fin du fichier et retourne dans `processes_loaded` le nombre total de processus chargés.

## 1.2 Fonctions Bonus

```
void processus_print(t_processus *process);
```

Affiche une chaîne de processus de manière formatée.

# 2 Fonctions FIFO

## 2.1 Structures

```
typedef struct s_FIFO
{
    t_processus *first;
    t_processus *last;
    int size;
} FIFO;
```

Cette structure encapsule une chaîne de processus, permettant un coût d'enfilage et de défilage en O(1).

```
typedef enum PROCESSFIELDS
{
    PID,
    ARRIVEE,
    DUREE
} PROCESSFIELDS;
```

On ajoute une énumération représentant les attributs d'un `t_processus`. La fonction sera utilisée dans les fonctions de tris pour pouvoir undiquer l'attribut en fonction duquel on trie les processus.

## 2.2 Init et Clear

Les fonctions :

```
t_processus* fifo_init ();
t_processus* fifo_clear (t_processus* file);
```

deviennent :

```
FIFO *fifo_init();
FIFO *fifo_init_from_process(t_processus *process);
FIFO *fifo_init_sorted_from_process(t_processus *process, PROCESSFIELDS field);
void fifo_free(FIFO **queue);
```

Deux fonctions permettent d'initialiser une FIFO à partir d'une chaîne de processus existante : l'une conserve l'ordre initial, l'autre trie selon un donné `PROCESSFIELDS field`.

`fifo_free` reçoit un `FIFO **` pour mettre le pointeur à `NULL`.

## 2.3 Taille

```
int fifo_vide (t_processus* file);
```

devient :

```
int fifo_is_empty(FIFO *queue);
```

Cette vérification devient triviale grâce à la structure FIFO.

## 2.4 Empilage et dépilage

```
t_processus* fifo_enfiler (t_processus* file, t_processus* p);
t_processus* fifo_defiler (t_processus** file);
```

deviennent :

```
void fifo_add(FIFO *queue, t_processus *process);
void fifo_add_sorted(FIFO *queue, t_processus *process, PROCESSFIELDS field);
t_processus *fifo_unqueue(FIFO *queue);
```

On choisi de réutiliser les FIFO dans l'implémentation de listes triées, ce qui veut dire que si on insère une chaine de processus dans une FIFO en utilisant uniquement la fonction `fifo_add_sorted` avec le même `PROCESSFIELDS field`, on se retrouve avec une FIFO triée par l'attribut spécifié dans `field`. Nous n'avons pas implémenté de tri 'descendant', car il n'est pas utilisé dans le sujet (donc lorsque référence est faite à une liste triée, cette liste est triée de manière 'ascendante')

## 2.5 Fonctions Bonus

```
void fifo_print(FIFO *queue);
int fifo_is_sorted(FIFO *queue, PROCESSFIELDS field);
```

On ajoute une manière de pouvoir visualiser les FIFO de manière formatée On ajoute également une fonction pour vérifier si une liste est triée selon un attribut `PROCESSFIELDS field` particulier.

# 3 Fonctions de simulation

```
void simuler_fcfs(t_processus* tableau, int nb_processus);
void simuler_sjf(t_processus* tableau, int nb_processus);
```

deviennent :

```
void simuler_fcfs(FIFO *tab);
void simuler_sjf(FIFO *tab);
```

Grâce à la FIFO, il n'est plus nécessaire de fournir le nombre de processus : la taille est stockée dans la structure.

## 3.1 FCFS

```
void simuler_fcfs(FIFO *tab);
```

## 3.2 SJF

```
void simuler_sjf(FIFO *tab);
```