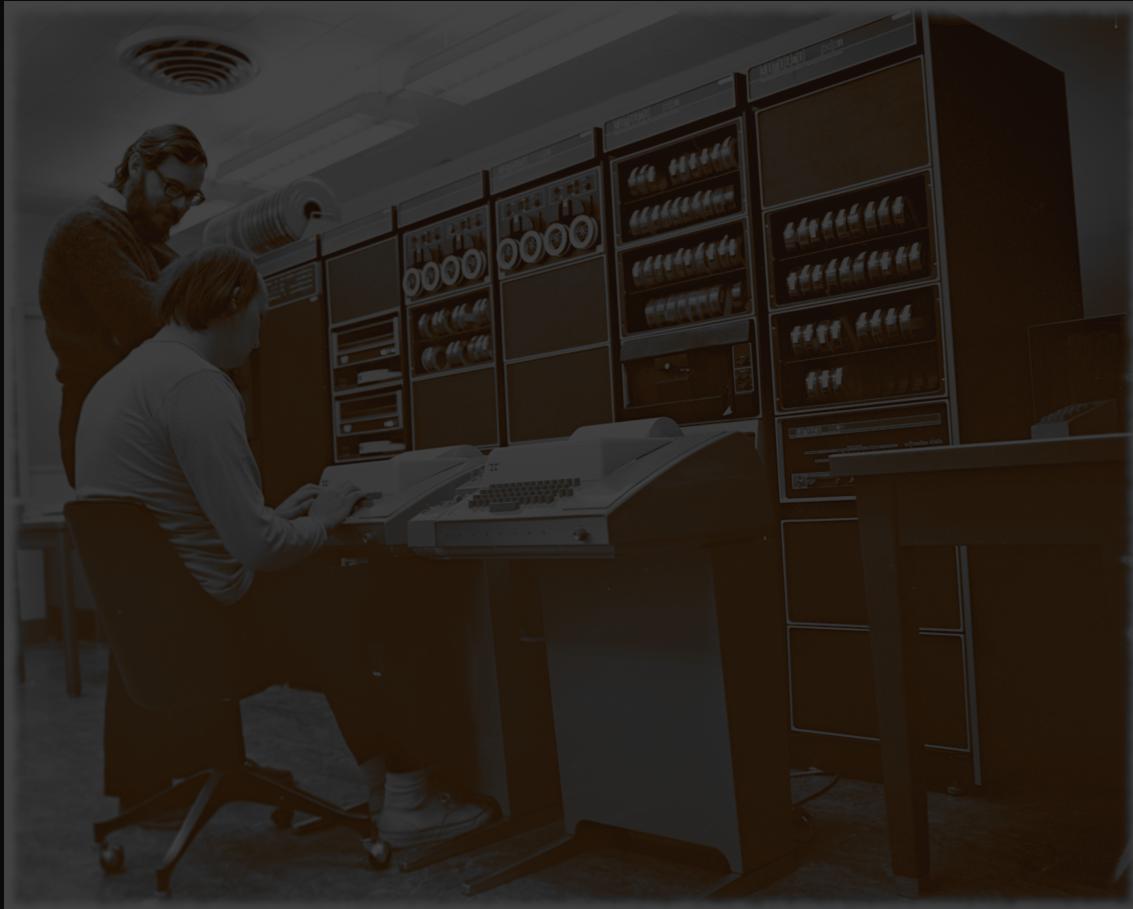


# Systèmes d'exploitation

TP : produit matriciel multithreadé

Alain Lebret

2024-2025



**Objectif**

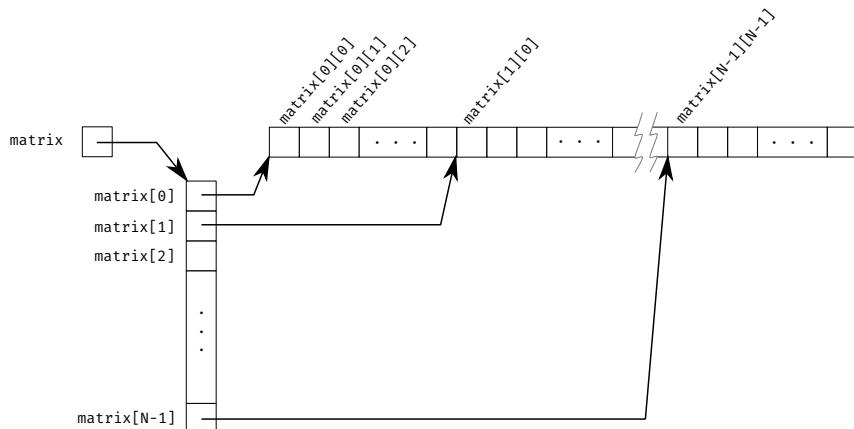
Mettre en oeuvre les *threads* sur Unix.

**Pré-requis** : chapitre sur les threads

**Durée estimée** : 1 séance

## Présentation

Dans cette séance, nous allons mettre en oeuvre les *threads* POSIX afin de réaliser des produits matriciels. La représentation des matrices en mémoire est indiquée sur la figure 1.



**Figure 1:** Représentation d'une matrice en mémoire.



La commande « `htop` » vous sera utile pour observer la manière dont le noyau du système d'exploitation gère la répartition des *threads* sur les coeurs du processeur.

## Travail à réaliser

Afin de simplifier le traitement, vous pourrez vous limiter à la multiplication de matrices carrées.

1. Vérifiez le nombre de coeurs  $N$  sur votre système en lançant la commande `lscpu` (section « Processeur(s) »).
2. Créez le sous-dossier « `matrix-product` » dans votre dossier de travail « `os` », puis copiez-y le contenu du fichier « `ressources.zip` » disponible sur la page de cours. Ouvrez un terminal dans « `matrix-product` ». L'arborescence devrait être la suivante :

```
.
├── Makefile
├── bin
└── include
    └── matrix.h
└── src
    ├── main.c
    └── matrix_product.c
```

3. Dans le fichier « `matrix.h` », modifiez la valeur des constantes `SIZE` à 10 et `PRINT` à 1 pour réaliser vos premiers essais. Produisez l'exécutable « `matrix_product` » en lançant la commande `make` et testez-le.
4. Le fichier « `matrix_product.c` » contient le code de la multiplication de deux matrices en mode mono-thread. Complétez-le afin qu'il réalise ce produit à l'aide de  $N$  threads,  $N$  correspondant au nombre de threads disponibles sur votre machine. Chaque thread s'occupera du calcul d'un  $N$ -ième de la matrice résultat.

Dans le cas où  $N = 4$ , vous pouvez découper la matrice résultat en 4 quadrants<sup>1</sup> comme le montre la figure 2, et associer chacun de ces quadrants à un *thread* spécifique. Pour transmettre à la fonction réalisée par chaque *thread* les informations sur les matrices en entrée, la matrice résultat et le quadrant que le *thread* devra traiter, vous pourriez utiliser dans ce cas-là une structure du type :

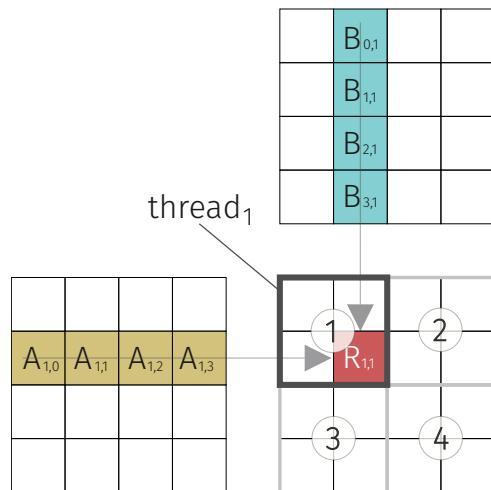
```
typedef struct args_t {
    int quadrant; /* quadrant sur lequel un thread donné va travailler */
    matrix_t *m1;
    matrix_t *m2;
    matrix_t *result;
} args_t;
```

Vous adapterez cette structure à votre cas particulier si vous utilisez votre machine personnelle. Testez votre programme et observez la charge des coeurs du processeur.

---

<sup>1</sup>Vous pouvez aussi choisir de « découper » la matrice résultat en  $N$  blocs de colonnes ou  $N$  blocs de lignes. L'important étant de répartir au mieux la charge de calcul entre les threads.

5. Pensez à remettre votre constante PRINT à 0, puis comparez les temps d'exécution avec plusieurs jeux de données de tailles différentes (10, 100, 500, 1000, 2000, 3000, 5000, 10000<sup>2</sup>). Dans quelles conditions peut-on espérer observer un gain de temps d'exécution ?



**Figure 2:** Association pour le *thread* n° 1 du quadrant à calculer.

## Livrable

Avant la date limite, déposez sur <https://foad.ensicaen.fr> l'archive compressée « `matrix-product-<votre nom>.zip` » qui contiendra le contenu de tout le dossier « `matrix-product` », excepté les fichiers objets et les exécutables.

Vous y intégrerez un fichier « `README.md` » qui permettra de :

- vous identifier ;
- décrire brièvement le contenu du dossier (objectif et composants) ;
- donner les instructions pour générer l'exéutable ou nettoyer l'arborescence.

## Résumé

Dans cette séance vous avez mis en place les *threads* POSIX de la bibliothèque *Pthread* afin de réaliser un produit matriciel.

<sup>2</sup>Attention, pour une matrice carrée de **10000 × 10000** le temps de calcul peut dépasser les 20 minutes sur des machines à 4 coeurs !