SCIENCES SORBONNE UNIVERSITÉ

PSAR
Mémoire partagée répartie

Maxime Derri
Damien Albisson
Changrui Zhang

26 mai 2023

Plan

1. **Objectif du projet.**
2. **Problématiques de conception.**
3. **Architecture.**
4. **Tests.**
5. **Conclusion et ouverture.**

Segment de mémoire partagée répartie ?

Segment de mémoire accessible depuis :

- Plusieurs processus sur une même machine.
- Plusieurs processus sur des machines distantes.

Cohérence forte sur la mémoire :

- Chaque processus sur un accès en lecture ou en écriture doit posséder la dernière version de celle-ci.

Eléments de base

- Maître et esclaves (modèle Client/Serveur).

- Le maître possède toujours la dernière version du segment de mémoire.

- Les segments de mémoire des esclaves ne sont pas forcément à jour, et ces derniers doivent faire des requêtes au maître pour le devenir.

Eléments de base (suite)

- Cohérence du segment de mémoire tenu page par page.

Plan

1. **Objectif du projet.**
2. **Problématiques de conception.**
3. **Architecture.**
4. **Tests.**
5. **Conclusion et ouverture.**

Gestion du protocole applicatif

- Définition d'un protocole entre le maître et les esclaves.

- Comment différencier les échanges ?

  En utilisant un code d'opération et des champs pré-définis pour chaque type de requête.

- Format des requêtes :

  [opcode,<champ_1>,...,<champ_N>]

Cohérence des données

- userfaultfd :

  Ajout récent dans le kernel.
  Peu de documentation au niveau des structures pour la communication avec le noyau (ioctl).

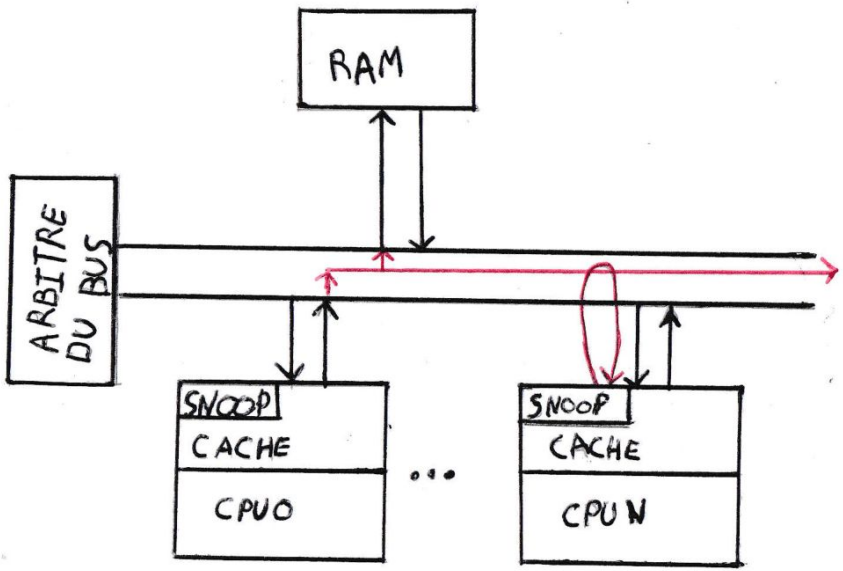
- Quand faire la mise à jour des pages pour les esclaves ?

  Au moment où ils vont vouloir accéder à cette page.

- Que faire en cas de modification sur la mémoire pour que les esclaves restent à jour ?

  Mécanisme d'invalidation des pages :
  inspiration du “snoop” pour la cohérence des caches matériels des coeurs sur architecture multi-coeurs.

Le snoop



Cohérence et accès concurrents

- Comment savoir qu'une page est à jour et qu'on peut y accéder ?

  Protections par mprotect (PROT_READ, PROT_WRITE, PROT_NONE).

- Comment gérer l'accès concurrent sur le segment de mémoire ?

  Implémentation de verrous sur les pages du maître pour l'exclusion lectures / écritures.

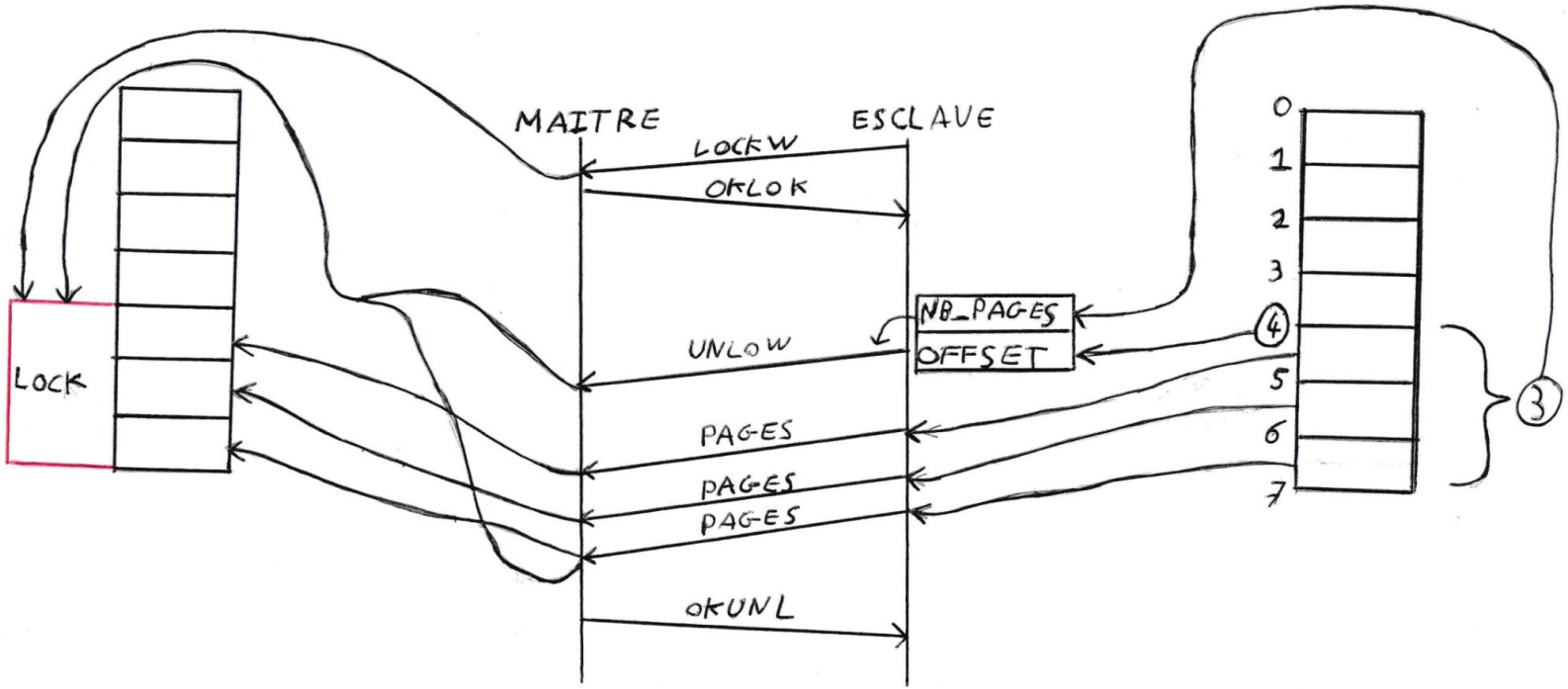
Plan

1. **Objectif du projet.**
2. **Problématiques de conception.**
3. **Architecture.**
4. **Tests.**
5. **Conclusion et ouverture.**

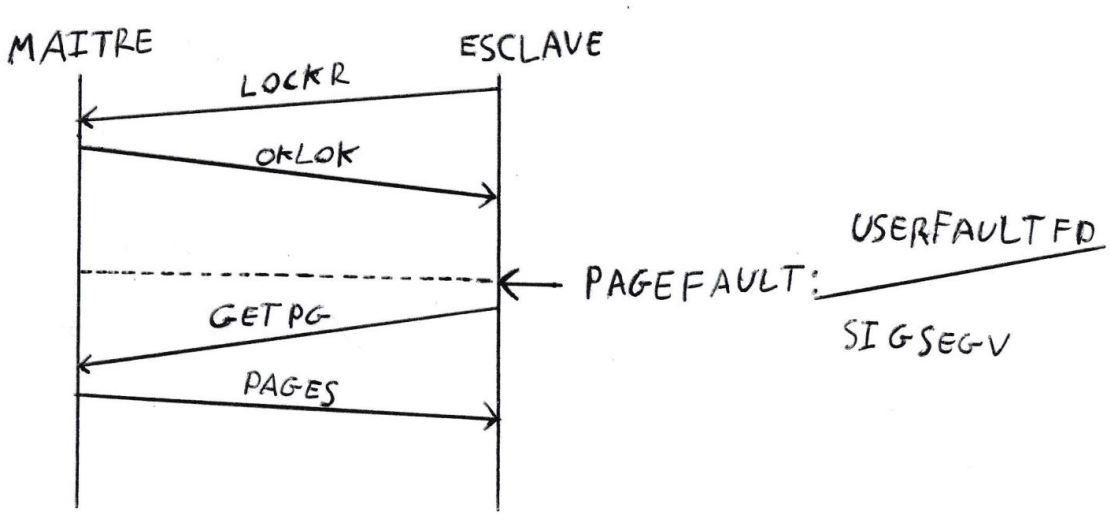
Gestion de la mémoire par pages

- Utilisation de userfaultfd pour le premier accès (page non mappée).
- Utilisation de mprotect et du signal SIGSEGV pour les autres accès.
- Une invalidation de page == mprotect (pour se remettre à jour par la suite).

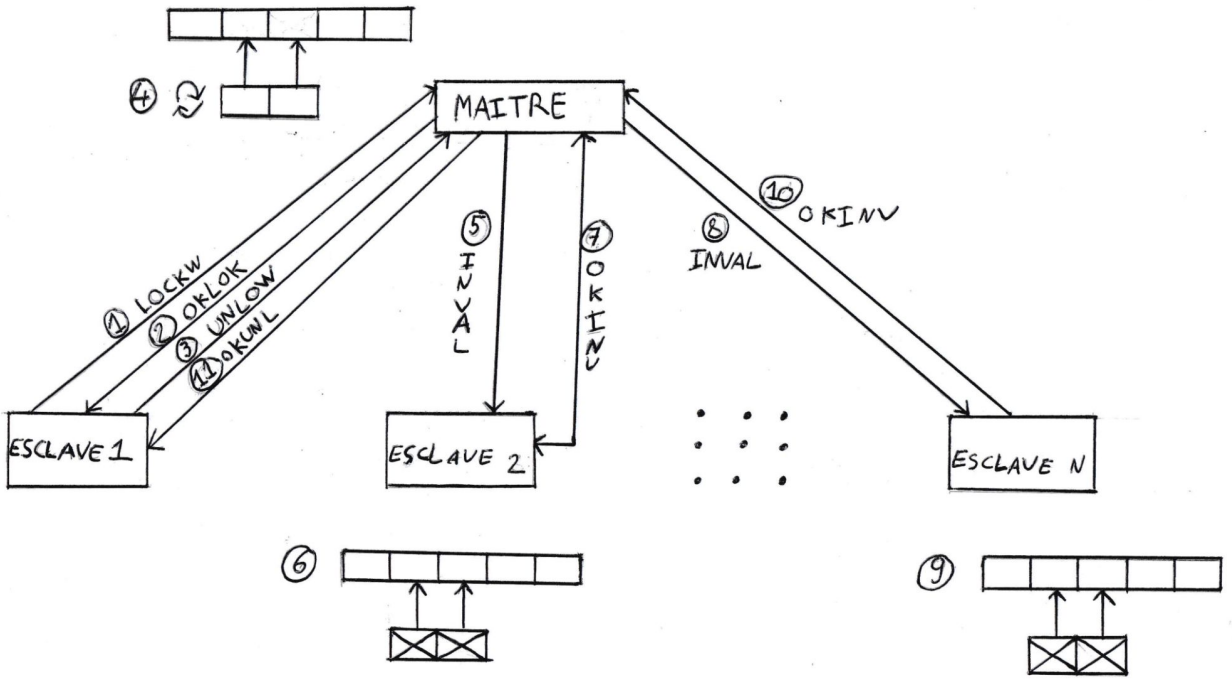
Mécanisme de mise à jour pour le maître



Mécanisme de mise à jour pour une page de l’esclave

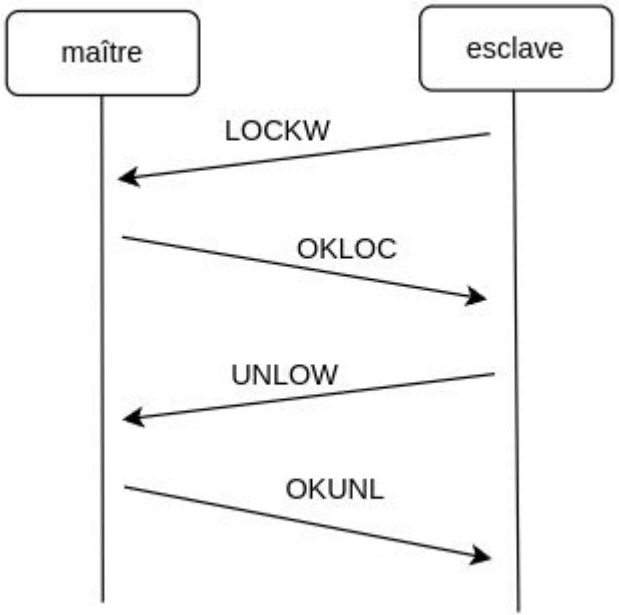


Mécanisme d'invalidation d'une page



Lock / Unlock

Socket TCP

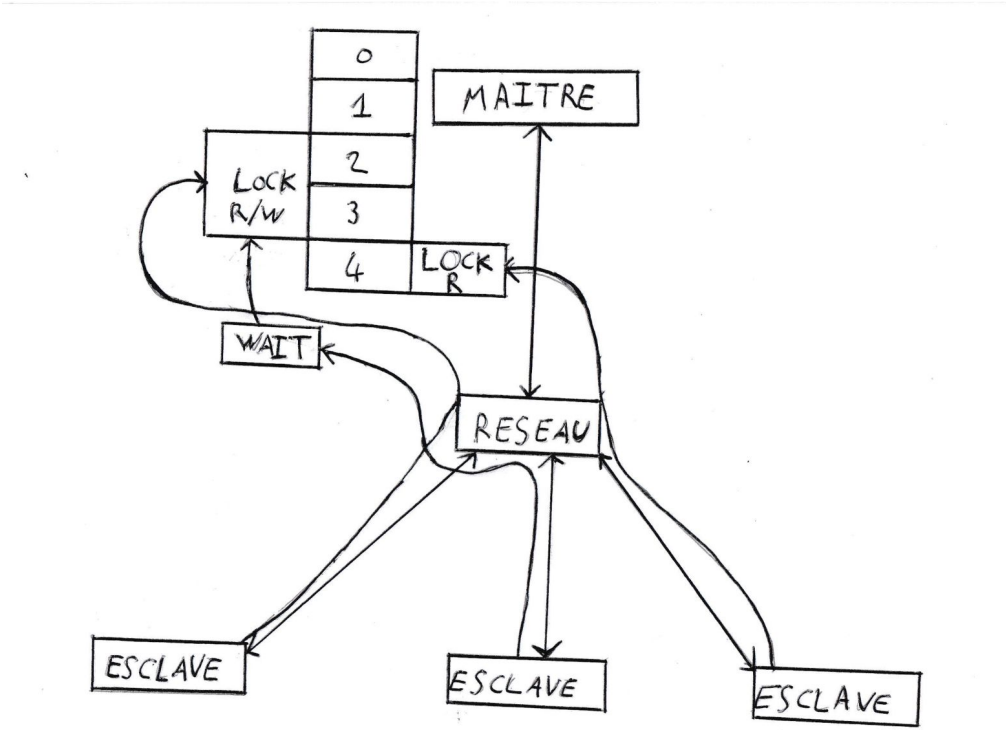


réponses :
- OKLOC
- OKUNL
- ERROR

verrouillage :
- LOCKR|offset|size
- LOCKW|offset|size

déverrouillage :
- UNLOR|offset|size
- UNLOW|offset|size

Gestion de la concurrence : lock sur la même page



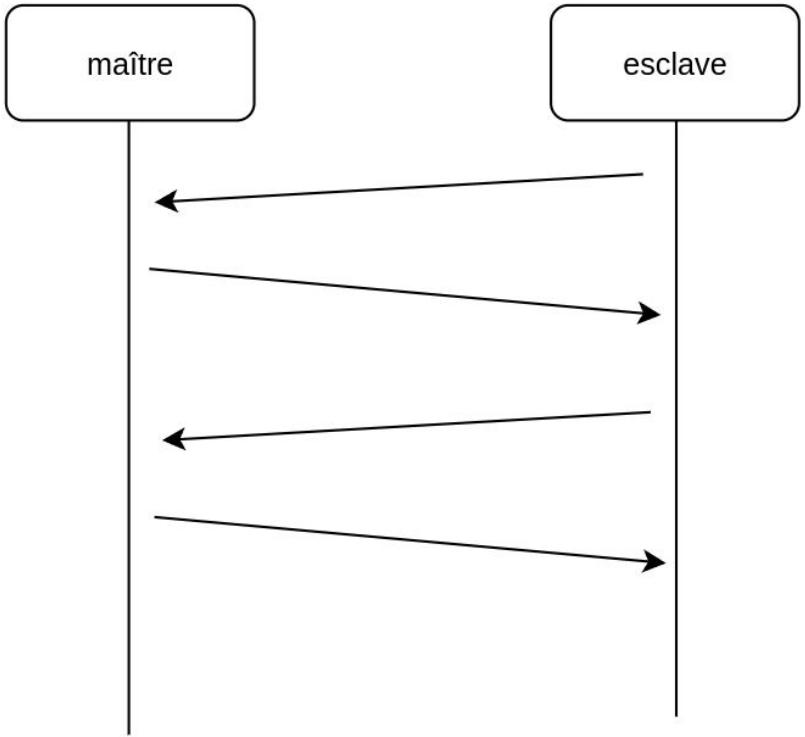
Autres protocoles applicatif

Connexion :
JOIN!|size

Déconnexion :
DECO!
Fin :
RELEA

Gestion des pages :
INVAL|offset|nbr_page
PAGES|offset|données

Erreur :
ERROR



Connexion :
JOIN?

Déconnexion :
DECO?

Gestion des pages :
OKINV
PAGES|offset|données
GETPG|offset

Initialisation et scrutation des requêtes par le maître

- Initialisation du maître, préparation du point de connexion (socket TCP) et écoute des requêtes des esclaves au cas par cas (dispatcher les tâches sur des threads).

- Des structures représentant les esclaves sont stockées dans une liste pour garder des informations sur l'état des esclaves connectés au maître.

- A la déconnexion (ou détection d'un problème avec les codes de retour) d'un esclave, la structure associée sera retirée de cette liste.

Arrêt du maître

- Le maître exécute la fonction loop_master pour écouter et répondre aux requêtes (comme vu précédemment).

- On peut terminer le maître proprement par la redéfinition d’un handler pour le signal SIGINT.

- Quand le maître décide de se terminer, il prévient chaque esclave.

Plan

1. **Objectif du projet.**
2. **Problématiques de conception.**
3. **Architecture.**
4. **Tests.**
5. **Conclusion et ouverture.**

Programmes de test

- Suite de Fibonacci : accès concurrent sur une page.
- page_concurrence : accès concurrent sur une page.
- diff_page_acces : accès indépendants sur différentes pages.
- test_lecture_ecriture : cohérence lecture écriture.

Suite de Fibonacci

Objectif : montrer que l'accès à la page est concurrent.

$$f(n) = f(n-1) + f(n-2)$$

```c
struct fibonacci{
    uint64_t N;
    uint64_t f0;
    uint64_t f1;
    uint64_t iter;
};
```

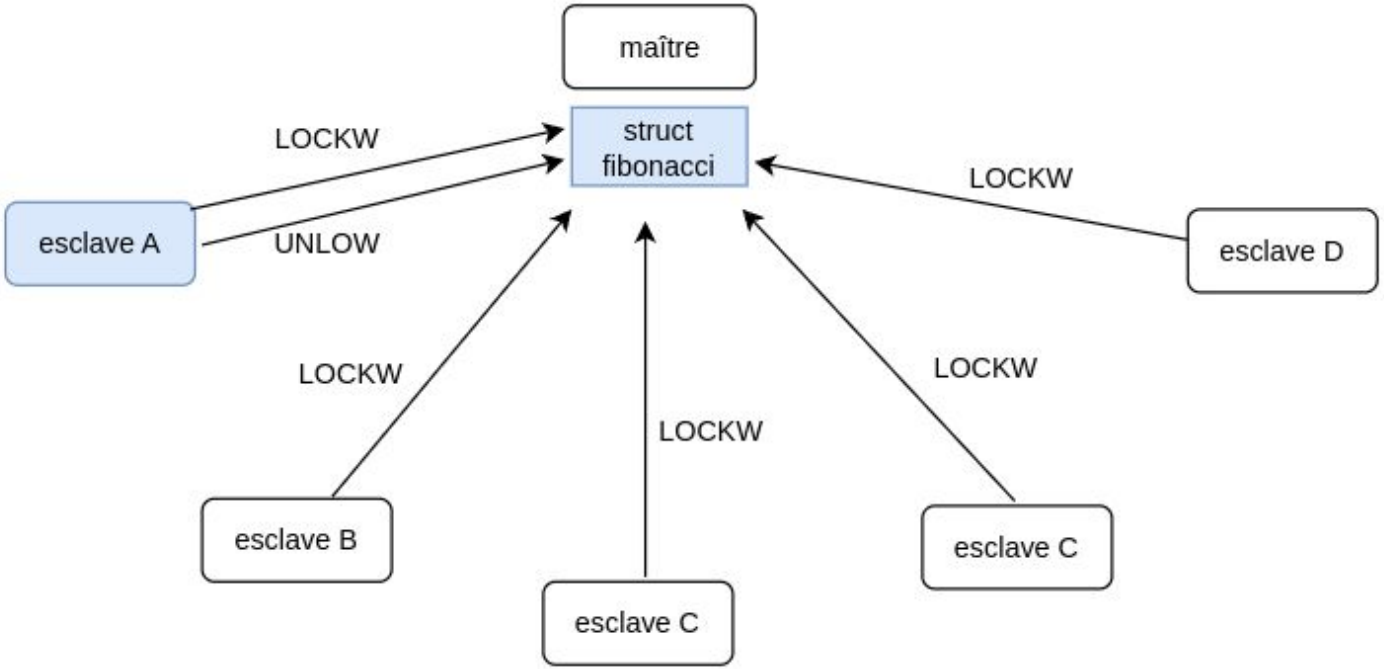
maître

```c
int main(){
    struct fibonacci *fibo;
    fibo = (struct fibonacci*) InitMaster(sizeof(struct fibonacci));
    /* Initialisation du segment */
    fibo->N = 20;
    fibo->f0 = 0;
    fibo->f1 = 1;
    fibo->iter = 1;
    loop_master();
    return 0;
}
```

esclaves (5)

```c
while(1){
    lock_write(fibo,sizeof(fibo));
    if(fibo->N <= fibo->iter + 1){
        //condition d'arrêt
    }
    //increment iteration
    fibo->iter++;
    //calcule
    uint64_t f0 = fibo->f0;
    uint64_t f1 = fibo->f1;
    uint64_t f2 = f0 + f1;
    //mise à jour struct fibonacci
    fibo->f0 = f1;
    fibo->f1 = f2;
    unlock_write(fibo, sizeof(fibo));
}
```

Calculer suite de Fibonacci pour N = 20



Résultat de Fibonacci

```
*************************
pid:89867
f0:4181
f1:6765
N:20
iter:20
f(20) = 6765
*************************
```

5 esclaves affichent
le résultat.

```
*************************
pid:89869
f0:4181
f1:6765
N:20
iter:20
f(20) = 6765
*************************
```

```
*************************
pid:89866
f0:4181
f1:6765
N:20
iter:20
f(20) = 6765
*************************
```

```
*************************
pid:89868
f0:4181
f1:6765
N:20
iter:20
f(20) = 6765
*************************
```

```
*************************
pid:89870
f0:4181
f1:6765
N:20
iter:20
f(20) = 6765
*************************
```

Plan

1. **Objectif du projet.**
2. **Problématiques de conception.**
3. **Architecture.**
4. **Tests.**
5. **Conclusion et ouverture.**

Conclusion

- Le maître traite les requêtes des esclaves une par une en respectant notre protocole.

- Lecture/écriture impossible sur une même page au même moment.

- Lecture/écriture sur différentes pages simultanées possible.

Pistes d’amélioration

- Gestion des pages de taille différente :
  - prendre en compte la taille de chaque entité sur le système ?
  - taille de “page” globale ? Problème avec l’utilisation de userfaultfd…

- Plusieurs lecteurs simultanés :
  - utiliser des pthread_cond_t et des compteurs.

- Patch kernel pour userfaultfd :
  - peut être complexe.
  - portabilité ?
  - maintenabilité ?