Rapport Projet System M1 – 2017 Paris Diderot

ATOMIC CIRCULAR-BUFFER Message queue IMPLEMENTATIONS (UNIX)

Fonctionnalités

Le sujet de bases ainsi que les extensions proposées sont implémentés.
Le README détaille les fonctionnalités et implémentations proposé.
Les statistiques sont dans le ODS
À retenir le code est factorisé au maximum et est :

- Compliant
  On essaye de lire et écrire au maximum que le demande l'utilisateur tout en respectant les limite d'atomicité choisit et de mode de blocage de l'appel

- Fiable
  La valeur de retour est toujours testé (lock, unlock, wait, open …)

Concurrence

- La version SPSC est fonctionnelle.

- La version MPMC n'est pas terminé, se pose le problème de stocker efficacement les lectures et écritures qui ne sont **pas encore validés** (car nous maintenons l'ordre, c'est une queue).

Ces lecture et écritures (**pas encore validés**) qui se terminent avant leur précédente doivent :

- Soit attendre que leur précédente soit fini (cas non satisfaisant, équivalent à une SPSC).
- Soit noté leur travail accomplit sous forme d'un couple (le point de départ et de fin manipulé dans le conduit) dans une structure de données que nous appellerons **StructAValider**

Légende : Quand le conduit vient juste d'être crée : **écritureTMP** est égale à **ecriture**

- **ecriture** est le curseur de la position de la dernière écriture validé aux lecteurs.
- **écritureTMP** est le curseur de la position de la dernière écriture pas encore validé aux lecteurs

Exemple pour une écriture :

```
1   Je regarde si la position du curseur d'écritureTMP me permet de travaillé ;
2   Je déplace le curseur d'écritureTMP à la fin de ma zone de travail ;
4   Je travaille sur ma zone ;
5   IF ( écritureTMP == ecriture ){  // aucune écriture avant moi n'est pas fini
6         IF (  StructAValider isNotEmpty) { // des écriture pas valider qui ont fini avant moi
7               Je calcule la borne maximal de travaille contigüe à valider.
8               Je déplace  ecriture  à cette borne
9               Signal  StructAValider
10        }
11  } ELSE { // une écriture avant moi n'est pas fini
12        IF ( StructAValider isFULL){
13              wait ;  goto 5 ;
14        } ELSE {
15              J'inscris mon travail dans StructAValider
16        }
17  }
18  Je retourne la taille de mon travail
```

Cas Limite

Étant donné que le projet autorise des écritures et lectures inferieure à l'atomicité choisit par l'utilisateur, le cas limite d'un conduit de taille N d'atomicité N avec des lectures et écritures de taille 1 est *problématique* car dans le pire des cas où la première écriture avec une priorité faible était préempter et que les $N-1$ écritures à la suite sont faites par des threads de priorité haute, il faudrait stocker dans la **StructAValider** les $N-1$ écritures pas encore validées.

Taille StructAValider

Ainsi cette structure de donnée serait au minimum de taille $2 * N$ pour stocker chaque couple de curseur définissant la zone de travail (ce qui n'est pas élégant vis-à-vis d'un conduit de taille N)

Cette taille pourrait être limitée à $\sqrt{N}$ (N la taille du conduit)

Ainsi quand le curseur de début d'une écriture à valider serait supérieur à ce qu'il est stockable dans la structure, elle se mettrait en attente

Ainsi quel que soit la complexité de la structure de données elle serait amortie par cette borne

En pratique (avec le programme Julia) il y a rarement plus de 5 écritures ou lecteur en attente de validations

Complexité StructAValider

Le problème de choisir une structure de données réside dans l'insertion trié et l'itération efficace de ses insertions

- Liste chainée à insertion triée, sous cette forme

  Le calcul de la zone contiguë a validé par le thread 1 serait de cout max $2 * \mathrm{N}$

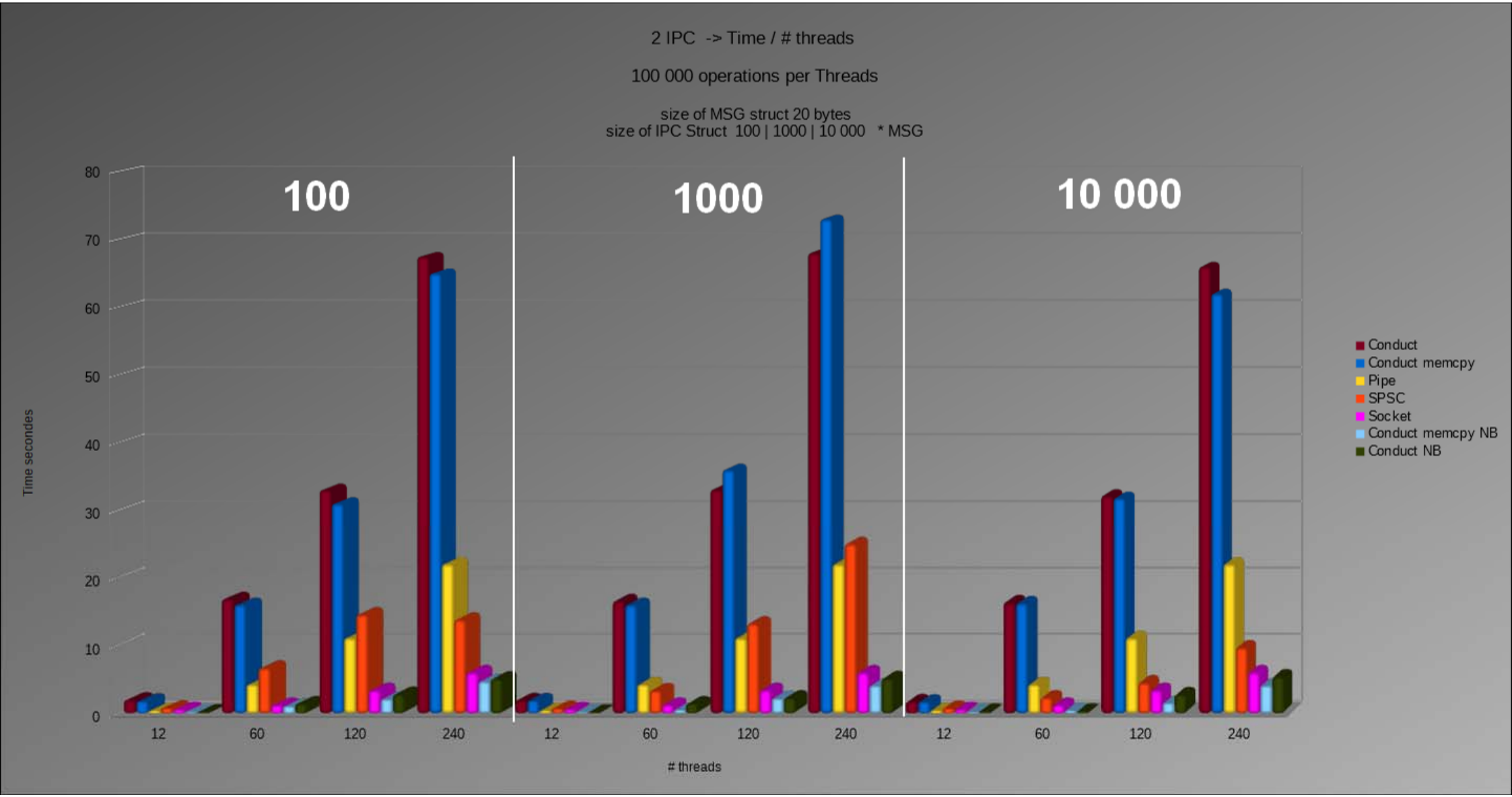
  Car si la zone contiguë à valider s'étend de la position N-1 dans le conduit (la dernière case) à l'avant dernière case (passe donc par la fin, nous sommes dans un conduit circulaire) ainsi il faut parcourir toute la structure pour trouver la dernières cases et ensuite encore tout un parcourt pour valider toutes les autres cases

  Exemple
  Nous somme la case bleu et nous devons valider les cases oranges (4 est en Head car il représente la position 0 du conduit mais est la 4 écriture à valider introduite dans la structure de donnée)

Head	4	5	6	6	7	8	9	10	11	12	13	14	15	16		1	2	3	Tail

  L'insertion des $\mathrm{N}-1$ écritures à valider serait de cout $\frac{N*(N+1)}{2} \approx N^2$

- Reste à trouver mieux où se satisfaire de cette implémentation


2 IPC -> Time / # threads
100 000 operations per Threads
size of MSG struct 20 bytes
size of IPC Struct 100 | 1000 | 10 000 * MSG
100
1000
10 000
Time secondes
0
10
20
30
40
50
60
70
80
12
60
120
240
threads
Conduct
Conduct memcpy
Pipe
SPSC
Socket
Conduct memcpy NB
Conduct NB