Université Paris Cité

Master 2 - Année 2023-24

# Programmation répartie

## TP2 -3 ème série

**Exercice 1.**— Montrer que le numéro de consensus d'un objet pile est 2.

**Exercice 2.**— On propose d'implémenter une pile d'objets de capacité $M\_SIZE$ avec un tableau partagé $Stack$ de $M\_SIZE$ éléments, chaque élément est un couple : valeur, compteur. Le haut de la pile (la variable partagée Top) est un triple index, valeur, compteur.
On utilise des compare_ and_ swap (V.C&S(b,c) signifie si la valeur qui est dans V est égale à b alors on affecte c à V et on retourne vrai sinon on retourne faux)
Initialement $Top$ est (0, Null, 0) et tous les éléments de $Stack$ sont initialisés à (Null,0) le pseudo code suivant.

```
push(v){
 {while (true){
   (index, value, counter)<--Top
   stackTop<--Stack[index].value
   Stack[index],C&S( (stackTop, counter-1), (value,counter))
   if (index=M_SIZE-1 ) then return Full
   aboveTop<-- Stack[index+1].counter
   if  Top.C&S(( (index, value, counter), (index+1, v, aboveTop+1)) then return ok
   }}}

pop(){
 {while (true){
   (index, value, counter)<--Top
   stackTop<--Stack[index].value
   Stack[index].C&S( stackTop,counter-1), ( value, counter))
   if (index=0) then return Empty
   belowTop<-- Stack[index-1]
   if  Top.C&S(  (index, value, counter), (index-1, belowTop.value,belowTop.counter+1)) then return valu
   }}}
```

1. A-ton besoin d'utiliser des compare_and_ swap pour implementer une pile?
2. L'implémentation réalise-t-elle une pile atomique? (Où sont les points de linéarisation?)
3. L'implémentation proposée est-elle non blocking? wait-free?
4. Implémenter cette pile en java à l'aide du package concurrent.

**Exercice 3.**— On a un objet $snapfaible$ muni de 2 opérations $write$ et $scan$. Sa spécification séquentielle est la même que celle du $snapshot$. Une thread qui l'utilise fait un write suivi d'un scan une seule fois.
Pour l'implémenter pour $n$ threads, on utilise un tableau partagé $tab$ de $n$ éléments.
Le pseudo code d'une thread $T_i$ est le suivant

```
void write (v){
variables locales :
    b=un booléen initialisé à vrai ;
    L=un ensemble initialisé à  {(v, i)};

   while (b)
     {for (i=0; i<n; i++){
         tab[i]=L;
         lire le tableau tab:
         ajoute les valeurs jamais vues à L
         si tous les éléments de tab sont égalux à  L alors b=faux
      }}}
T[]  scan() {
variables locales :
   b=un booléen initialisé à vrai ;
   L=un ensemble initialisé à vide;

   for (i=0; i<n; i++){
      L=L union {tab[i]\;
   while (b)
     {for (i=0; i<n; i++){
        tab[i]=L;
        lire le tableau tab:
         ajoute les valeurs jamais vues à L
         si tous les éléments de tab sont égalux à  L alors b=faux
      }}

      mettre dans un tableau à l'indice $i$ la valeur de $v$ si $(v, i)$ est present dans L;
      retourner ce tableau ;}
```

1. Si $n$ threads exécutent *write*, puis *scan*, est il possible qu'une thread ne termine pas?
2. On exécute 2 threads T1 et T2, T1 exécute $write(v1)$ et T2 $write(v2)$. Est il possible que T2 ne mette jamais $v1$ dans son ensemble L? Dans ce cas quelle valeur T1 a-t-elle dans son ensemble L?
3. On exécute 3 threads T1, T2 et T3 ou T1 exécute $write(v1)$, T2 $write(v2)$ et T3 $write(v3)$. Est il possible que ni T1 ni T2 ne mettent jamais la valeur de v3 dans leur ensemble L? Dans ce cas quelles valeurs T1 et T2 peuvent elles avoir dans leur ensemble L?
4. Si une thread qui exécute $write(v)$ termine à l'instant $t$, est ce que toutes les threads qui commencent après $t$ (qu'elles terminent ou pas) auront dans leur ensemble $L$ la valeur $v$?
5. Si on exécute $n$ threads quelle sont les valeurs possibles des ensembles L?
6. L'implémentation du $snapfaible$ proposée est-elle linéarisable ? est-elle wait free?
7. Implémenter cet objet en java.
8. Répondre à nouveau à la question 6 , si le tableau *tab* contient $n/2$ éléments ( $n > 6$) et que l'objet est utilisé par $n$ threads.