Algorithme de Lamport
Application à m producteurs / 1 consommateur

Contrôle du consommateur

```
Processus C :
     Entier debcons, fincons, ifinprod ;
     Booléen req_en_cours, sc_en_cours;

     (debcons, fincons, ifinprod) ← (0, 0, 0) ;
     (req_en_cours, sc_en_cours) ← (faux, faux) ;

     Tant que
     /* ACQUISITION */
     Soit ¬ req_en_cours et Cons ? besoin_sc() Alors
          req_en_cours ← vrai

     /* SECTION CRITIQUE */
     Soit req_en_cours et ¬sc_en_cours
       et debcons-ifinprod < 0 Alors
               debcons ← debcons + 1 ;
               Cons ! début_sc() ;
               sc_en_cours ← vrai

     /* LIBERATION */
     Soit req_en_cours et sc_en_cours et Cons ? fin_sc() Alors
               fincons ← fincons + 1 ;
               k ← 1 ;
               Tant que  k <  n + 1 Faire
                         P[k] !! màj(fincons) ;
                         k ← k + 1 ;
               Fin Tant que
               sc_en_cours ← faux ;
               req_en_cours ← faux

     /* réception de ifinprod */
     Soit P[j] ? màj(ifinprod) Alors
               rien
     Fin Tant que Soit
Fin Processus
```

Contrôle du producteur n°i

```
Processus P[i :1..m] :

     /* hl = heure locale
        he = heure externe (reçus dans une estampille) */
     Entier hl, he, debprod, finprod, ifincons ;

     /* tableau à m lignes (m = nombre de processus
        producteurs) d'éléments de type (type, date) */
     Table<({req, rel, ack}, entier)> tab[1..m] ;

     /* booléens pour modéliser l'état actuel du contrôleur */
     Booléen req_en_cours, sc_en_cours ;

     /* initialisation */
     (hl, he, debprod, finprod) ← (0, 0, 0, 0);
     tab[i:1..m] ←  (rel, 0) ;
     (req_en_cours, sc_en_cours) ← (faux, faux) ;

     /* procédure permettant de diffuser à l'ensemble des
        autres contrôleurs un message msg (hl, i). Ce message
        est de type req ou rel. */
     DIFFUSER (msg, hl, i) ;

     /* procédure permettant de mettre à jour l'horloge locale
        hl d'une date he reçue via une estampille */
     MAJ_H (hl, he) ;

     /* renvoie l'identifiant du processus ayant la plus
        vielle date dans le tableau tab */
     PLUS_VIEILLE_DATE(tab) ;

     Tant que
     /* réception d'un message de type req */
     Soit P[j] ? Req(he, j) Alors
               MAJ_H (hl, he) ;
               hl ← hl + 1 ;
               P[j] ! Ack(hl, i)
               tab[j] ← (req, he)

     /* réception d'un message de type ack */
     Soit P[j] ? Ack(he, j) Alors
               MAJ_H (hl, he) ;
               /* mise à jour du tableau sauf si le message
                  précédent de Pj est de type req */
               Si tab[j].type ≠ req Alors
                         tab[j] ← (ack, he)
```

```
                Fin Si

        /* réception d'un message de type rel */
        Soit P[j] ? Rel(he, j) Alors
                MAJ_H(hl, he) ;
                tab[i] ← (rel, he) ;
                /* mise à jour des compteurs debprod et finprod
                   car j a produit et a fini de produire */
                debprod ← debprod + 1 ;
                finprod ← finprod + 1

        /* ACQUISITION */
        Soit ¬ req_en_cours et Prod[i] ? besoin_sc() Alors
                hl ← hl + 1 ;
                req_en_cours ← vrai ;
                DIFFUSER(P[{1..m}\{i}], Req(hl, i)) ;
                tab[i] ← (req, hl)

        /* SECTION CRITIQUE */
        Soit req_en_cours et ¬ sc_en_cours
          et PLUS_VIEILLE_DATE(tab) = i
          et debprod – ifincons < n Alors
                debprod ← debprod + 1 ;
                Prod[i] ! début_sc() ;
                sc_en_cours ← vrai

        /* LIBERATION */
        Soit req_en_cours et sc_en_cours
          et Prod[i] ? fin_sc() Alors
                finprod ← finprod + 1 ;
                C !! màj(finprod) ;
                sc_en_cours ← faux ;
                hl ← hl + 1 ;
                DIFFUSER(P[{1..m}\{i}], Rel(hl, i)) ;
                tab[i] ← (rel, hl) ;
                req_en_cours ← faux

        /* réception de ifincons */
        Soit C ? màj(ifincons) Alors
                rien

        Fin Tant que Soit
Fin Processus
```