```
/*
  Exemple: Utilisation du parallelisme avec sac de taches pour
           le calcul de la somme de 1 a n.

  Arguments du programme:
   #1 (n) = valeur du n pour laquelle on veut calculer la somme
            1 + 2 + 3 + ... + n−1 + n
   #2 (k) = longueur de chacun des sous−intervalles pour lesquels
            une tache sera cree.

  Exemple, avec n = 1000, k = 25 => 40 taches (TOKEN) seront crees:
     1..25, 26..50, 51..75, ..., 951..975, 976..1000

  Note: Pour simplifier, n doit etre un multiple de k.

 */

#include "LOCK.h"

/* Procedure somme:
   Tache independante qui calculera la somme d'un intervalle
   commencant a borne_inf et de longueur nb.
 */
THREADED somme( int borne_inf,
                /* Nombre elements de l'intervalle. */
                int nb,
                /* Ou transmettre le resultat intermediaire. */
                int *GLOBAL resultat,
                /* Cadenas pour assurer l'acces exclusif au resultat. */
                LOCK resultat_barre,
                /* Signal pour indiquer la disponibilite du resultat. */
                SPTR resultat_disponible )
 {
  int k, res;

  /* On calcule la somme de l'intervalle de nombres. */
  res = 0;
  for ( k = borne_inf; k <= borne_inf + nb−1; k++ )
    res += k;

  /* On obtient l'acces exclusif a resultat avant de transmettre
     la valeur calculee. */
  LOCK_SYNC( resultat_barre, TO_SPTR(RESULTAT_BARRE) );

  FIBER RESULTAT_BARRE <* 1 *> {
    PUT_SYNC( res, resultat, resultat_disponible );
    TERMINATE;
  }
 }

void print_usage_and_die()
{
  printf( "usage:\n" );
  printf( "  sacs−avec−lock−2 n k\n" );
  printf( "  (n >= 1)\n" );
  printf( "  (k <= n)\n" );
  printf( "  (n % k == 0)" );
  exit(−1);
}

/*
```



```
*/
THREADED MAIN( int argc, char* argv[] )
{
  /* Note: les deux premieres variables contiendront les arguments
     du programme. */
  int n;          /* Nombre d'elements a additionner. */
  int k;          /* Longueur des intervalles pour chaque tache. */

  int tot;        /* Total cumulatif. */
  int v;          /* Resultat intermediaire. */
  LOCK resultat_barre;
                  /* Cadenas pour acces exclusif au resultat intermediaire. */
  int i;

  /* On lit les arguments. */
  if ( argc <= 2 )
    print_usage_and_die();
  n = atoi(argv[1]);
  if ( n < 1 )
    print_usage_and_die();
  k = atoi(argv[2]);
  if ( (k > n) || (n % k != 0) )
    print_usage_and_die();

  tot = 0;
  /* Initialisation: la sync slot FIN doit etre initialisee explicitement
     avec une instruction INIT_SLOT: la forme <* XXX *> n'est possible
     que si l'argument est une expression pouvant etre evaluee de
     facon statique. */
  INIT_SLOT( FIN, n/k );
  /* On cree le cadenas. */
  INIT_LOCK( &resultat_barre, TO_SPTR(CADENAS_CREE) );

  FIBER CADENAS_CREE <* 1 *> {
    /* On cree les divers processus iteratifs. */
    for( i = 0; i < (n/k); i++ )
      TOKEN( somme, i*k+1, k,
             TO_GLOBAL(&v), resultat_barre, TO_SPTR(CUMULER_SOMME) );
  }

  /* Fibre qui recoit et cumule les resultats intermediaires. */
  FIBER CUMULER_SOMME <* 1 *> {
    /* Un resultat intermediaire est disponible dans v. */
    tot += v;
    UNLOCK( resultat_barre );
    SYNC( FIN );
  }

  FIBER FIN {    /* Initialisation de la sync slot faite plus haut. */
    printf( "somme(%d) = %d\n", n, tot );
    TERMINATE;
  }
}
```