

**Contrôle continu - 23/10/2020****Durée : 1h30****Cours et/ou travaux dirigés autorisés.****Barème donné à titre indicatif : Ex.1 : 4 - Ex. 2 : 4 - Ex. 3 : 8 - Ex 4 : 4****Exercice 1.** Synchronisation des processus

Soient trois processus concurrents  $P_1$ ,  $P_2$  et  $P_3$  qui partagent les variables  $n$  et  $out$ . Pour contrôler les accès aux variables partagées, un programmeur propose les codes suivants :

Semaphore mutex1 = 1 ;

Semaphore mutex2 = 1 ;

Code du processus $P_1$	Code du processus $P_2$	Code du processus $P_3$
P(mutex1) ; P(mutex2) ; $out = out + 1$ ; $n = n - 1$ ; V(mutex2) ; V(mutex1) ;	P(mutex2) ; $out = out - 1$ ; V(mutex2) ;	P(mutex1) ; $n = n + 1$ ; V(mutex1) ;

Cette proposition est-elle correcte ? Justifier votre réponse.

Si elle n'est pas correcte, indiquer parmi les 4 conditions requises pour réaliser une exclusion mutuelle correcte, celles qui ne sont pas satisfaites ? Proposer une solution correcte.

Cette proposition n'est pas correcte : si  $P_2$  est en section critique et  $P_1$  a exécuté P(mutex1) alors  $P_1$  est bloqué et empêche  $P_3$  d'entrer en section critique.

Conditions non vérifiées : un processus en dehors de sa section critique bloque un autre processus. (ici  $P_1$  bloque  $P_3$ ). Pour corriger, on propose la version suivante du code de  $P_1$  :

```
P(mutex1) ;
n=n-1 ;
V(mutex1) ;
P(mutex2) ;
Out = out +1 ;
V(mutex2) ;
```

**Exercice 2.** Considérons un système de gestion de fichiers où le disque est géré à l'aide d'une FAT (File Allocation Table). La FAT est un tableau d'entiers qui possède autant de lignes qu'il y a de blocs sur le disque. Un bloc  $i$  est libre si  $fat[i] = \text{FAT\_FREE}$ . Si le bloc  $i$  est le dernier bloc d'un fichier, on a  $fat[i] = \text{FAT\_EOF}$ . Toute autre valeur positive sur  $fat[i]$  indique le bloc suivant. Écrire la fonction d'allocation qui recherche un espace libre de  $n$  blocs consécutifs et renvoie l'adresse du premier bloc (ou -1 en cas d'échec) en se basant sur le principe de meilleur ajustement (best-fit).

```
alloc_bloc_best_fit (entier[] fat, entier n) retourne entier
```

- une procédure auxiliaire qui retourne une zone qui convient à partir de l'indice  $i$  :  

```
procedure suitable_zone(entier[] fat, entier n, entier i) retourne entier * entier
var début, taille : entier
début <- i
taille <- 0
```

```

tantQue (i < fat.taille et fat[i] = FAT_FREE) faire
    i <- i+1
    taille <- taille + 1
finTQ
Si (taille >= n) alors retourne (début, taille) fsi
Si (i >= fat.taille) alors retourner (-1, 0) fsi
retourner suitable_zone(fat, n, i+1)

```

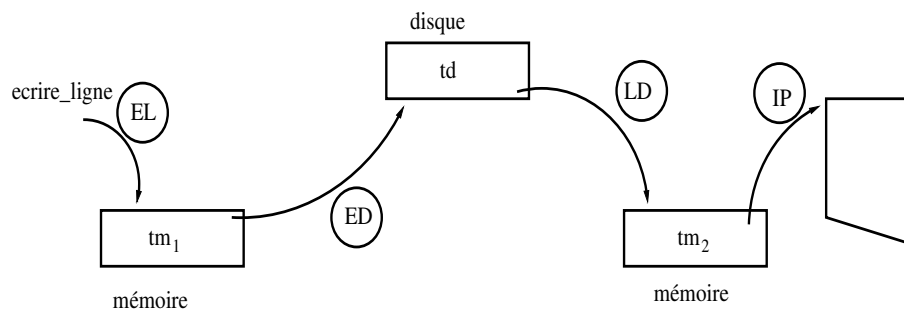
- la fonction demandée (dans une version non optimisée) :

```

procédure alloc_bloc_best_fit(entier[] fat, entier n) retourne entier
(suitable_adr, suitable_taille) = suitable_zone(fat, n, 0)
Si (suitable_adr = -1) alors retourner -1 fsi
Si (suitable_taille = n) alors retourner suitable_adr fsi
indice <- suitable_adr + suitable_taille
répéter
    (adr, taille) <- suitable_zone(fat, n, indice)
    Si (adr != -1) alors
        Si (taille < suitable_taille) alors
            // on change le suitable
            suitable_adr <- adr
            suitable_taille <- taille
        fsi
    fsi
    indice <- adr + taille
jusqu'à (indice >= fat.taille)
retourner suitable_adr

```

**Exercice 3.** Soit le spouleur d'impression schématisé comme suit :



où EL, ED, LD et IP sont quatre activités autonomes car elles s'exécutent sur des processus distincts. De plus, la procédure EL (*ecriture\_ligne*) est une procédure utilisateur.

On pose les hypothèses suivantes :

1. le transfert se fait par blocs de taille fixe, égale à celle d'une ligne. Chaque tampon  $tm_1$ ,  $tm_2$  et  $td$  est respectivement géré par un moniteur  $tampon_1$ ,  $tampon_2$  et  $tampon_{disc}$  de même structure qui a pour rôle
  - d'assurer l'exclusion mutuelle au tampon,
  - d'assurer la synchronisation sur les conditions *tampon plein* et *tampon vide*
2.  $tm_1$ ,  $tm_2$  et  $td$  sont de taille respective  $N_1$ ,  $N_2$  et  $N_{disc}$ , les tailles sont exprimées en nombres de blocs,
3. dans le tampon  $tampon_{disc}$ , les opérations de dépôt et de retrait (**entrer**, **sortir**) sont des E/S qui utilisent le moniteur de gestion du disque,
4. le système tout entier fonctionne en régime permanent sans limitation de lignes à imprimer et les séquences de traitement des erreurs sont omises.

### Questions :

1. Écrire la structure du moniteur  $tampon_1$  en supposant qu'on dispose des procédures **entrer(l:ligne)** et **sortir()** retourne ligne.  
(Donner le code des procédures **déposer(b:ligne)** et **retirer()** retourne ligne et préciser les variables et la (les) condition(s) du moniteur.)

```
moniteur tampon1
  var n : 0..N1
      non_vide, non_plein : condition
      // Déclaration des procédures
      procédure déposer(b : ligne)
        si n = N1 alors non_plein.attendre fsi
        n <- n + 1
        entrer(b)
        non_vide.signaler
      fin
      procédure retirer() retourne ligne
        Si n=0 alors non_vide.attendre fsi
        l <- sortir()
        n <- n - 1
        non_plein.signaler
        retourner l
      fin
  début
    n <- 0
  fin
```

2. En définissant  $tm_1$ ,  $tm_2$  et  $td$  comme des tableaux d'éléments de la taille d'une ligne, et des pointeurs *tête* et *queue* locaux à chaque moniteur, écrire les procédures **entrer** et **sortir**

a) dans le moniteur  $tampon_1$ ,

```
procédure entrer(l : ligne)
  tm1[queue] <- l
  queue <- queue + 1 mod N1
fin
procédure sortir() retourne ligne
  var l : ligne
  l <- tm1[tête]
  tête <- tête + 1 mod N1
  retourner l
fin
```

- b) dans le moniteur `tampondisc` pour lequel les opérations de dépôt et de retrait sont des E/S qui utilisent la primitive `disque.écrire(source, destination)` du moniteur de gestion du disque.

```
procédure entrer(l : ligne)
    disque.écrire(l, td[queue])
    queue <- queue + 1 mod Ndisc
fin
procédure sortir() retourne ligne
    var l : ligne
    disque.écrire(td[tête], l)
    tête <- tête + 1 mod Ndisc
    retourner l
fin
```

`tête` pointe sur la prochaine ligne à prélever et `queue` sur la première place libre dans le tampon.

3. Écrire les processus IP, ED, LD et la procédure utilisateur EL. Le processus IP utilisera la procédure `impr.écrire(l)` du moniteur de gestion de l'imprimante.

```
processus IP
    var l : ligne
    répéter
        l <- tampon2.retirer()
        impr.écrire(l)
    jusqu'à faux
fin
processus ED
    var l : ligne
    répéter
        l <- tampon1.retirer()
        tampondisc.déposer(l)
    jusqu'à faux
fin
processus LD
    var l : ligne
    répéter
        l <- tampondisc.retirer()
        tampon2.déposer(l)
    jusqu'à faux
fin
procédure EL(l : ligne)
    tampon1.déposer(l)
fin
```

**Exercice 4.** Dans  $r_0$  se trouve l'adresse d'une chaîne de caractères  $s$  terminée par l'octet nul et composée de mots. Un mot est un ensemble de caractères ne comprenant pas le caractère espace (32 en décimal). On vous demande d'indiquer ce que calcule ce programme et de préciser où se

trouve le résultat de ce calcul.

```
        MVI r1, #0
loop1 : LDB r2, (r0)
        ADD r0, r0, #1
        JZ r2, fin
        SUB r31, r2, #32
        JZ r31, loop1
        ADD r1, r1, #1
loop2 : LDB r2, (r0)
        ADD r0, r0, #1
        JZ r2, fin
        SUB r31, r2, #32
        JNZ r31, loop2
        JMP loop1
fin :
```

Le programme compte les mots de la chaîne `s` dont l'adresse est dans le registre `r0`. À la fin de l'exécution du programme, ce nombre de mots est dans le registre `r1`.