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. Considérons le programme suivant, qui se termine parfois par un interblocage :

Conditions initiales
a=1 b=1 c=1

Processus 1
P(a); P(b); V(b); P(c); V(c); V(a);

Processus 2
P(c); P(b); V(b); V(c); P(a); V(a);

Processus 3
P(c); V(c); P(b); P(a); V(a); V(b);

1. Listez toutes les paires de sémaphores que chaque processus cherche à obtenir.
   P1 : `(a, b)` puis `(a, c)` mais jamais `(b, c)`.
   P2 : `(c, b)` mais jamais les autres.
   P3 : `(b, a)`.
2. On cherche à éviter les interblocages en ordonnant les réservations selon l'ordre `a` < `b` < `c`. Discuter les réservations de chaque processus selon ce critère.
   P1 fait toutes ses réservations dans l'ordre, pas de problème.
   P2 viole l'ordre pour `(b, c)` mais aucun autre processus n'a besoin de cette paire dans cet ordre, donc c'est bon.
   P3 viole cet ordre pour `(c, b)` mais c'est bon car il ne les verrouille pas en même temps. Par contre, il viole aussi l'ordre pour `(a, b)` et ce n'est pas bon parce que P1 a aussi besoin de cette paire.
3. On sait que lorsque les requêtes sont ordonnées, il n'y a pas d'interblocage. En modifiant seulement le code de P3, proposer un ordre de réservation des requêtes `a`, `b` et `c` pour tous les processus qui garantit alors l'absence d'interblocage.
   Si on inverse `P(b)` et `P(a)` dans P3, c'est bon car tous respectent alors l'ordre `acb`. Si tous respectent un même ordre, alors il n'y a pas d'interblocage.

Exercice 2. Considérons le problème du barbier qui s'énonce comme suit : *La boutique du barbier est composée d'une salle d'attente contenant* `n` *chaises et du salon où se trouve la chaise du barbier. Lorsque le barbier a fini de raser un client, il fait entrer le client suivant dans le salon. Si la salle d'attente est vide, le barbier s'y installe pour dormir. Si un client trouve le barbier endormi, il le réveille. Si non, il s'installe dans la salle d'attente s'il reste de la place. S'il n'y a pas de place, il rentre chez lui.*

Considérons une solution pour ce problème qui assure que le client ne s'assoit pas sur le siège tant que le barbier n'est pas prêt :

```
INITIALISATION
Sémaphores clients, barbier, exclusion
clients = 0  // clients dans la salle d'attente
barbier = 0  // barbier prêt à couper
exclusion = 1 // pour l'exclusion mutuelle
int places = nb_chaises
```

```
     BARBIER
     while (true) {
(1)     P(clients)
(2)     P(exclusion)
(3)     places = places + 1
(4)     V(barbier)
(5)     V(exclusion)
(6)     coupe_cheveux
}
```

```
     CLIENT
(1)  P(exclusion)
(2)  si (place > 0) alors
(3)     places = places - 1
(4)     V(clients)
(5)     V(exclusion)
(6)     P(barbier)
(7)     se_faire_couper_les_cheveux
(8)  sinon
(9)     V(exclusion)
(10)    rentrer_chez_soi_les_cheveux_longs
(11) fsi
```

1. Quel schéma de synchronisation vu en cours est étendu ici ?
   C'est le schéma du producteur/consommateur avec plusieurs producteurs.
2. Dans quelle mesure peut-on permuter les instructions (4) et (5) dans le code de `BARBIER` ?
   On peut toujours permuter deux V dans un code sans changer la sémantique.
3. Même question si on permute les instructions (5) et (6) dans le code de `CLIENT` ?
   Si on place `V(exclusion)` après le `P(barbier)`, ça implique que les autres clients ne peuvent pas tester s'il reste de la place dans la salle d'attente (et y entrer) avant que ce client ne se soit fait couper les cheveux. Donc, la file d'attente se fait sur le sémaphore `exclusion`, i.e. la salle est vide et la queue est sur le trottoir.

Exercice 3. On se place dans un système de mémoire de 1700 Ko de mémoire haute (i.e. au-delà de la partie utilisée par l'OS) répartie en 5 partitions de 100 Ko, 500 Ko, 200 Ko, 300 Ko et 600 Ko.

On suppose que le système d'exploitation doit allouer des processus de taille 212 Ko, 417 Ko, 112 Ko et 426 Ko dans cet ordre. Une partition ne doit être occupée que par un seul processus. Pour chacun des algorithmes suivants, donner l'allocation obtenue et le taux de fragmentation : (1) prochain bloc libre, (2) meilleur ajustement, (3) plus grand bloc libre.

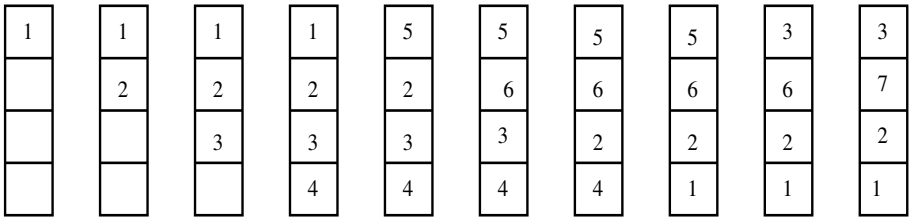
- First fit : P1(212), P2(417), P3(112) et P4(426).
  État initial : 100 500 200 300 600
  Placement de P1 : 100 P1+288 200 300 600
  Placement de P2 : 100 P1+288 200 300 P2+183
  Placement de P3 : 100 P1+288 P3+88 300 P2+183
  Placement de P4 : impossible tant que P1 ou P2 n'a pas libéré sa zone.
- Best fit : P1(212), P2(417), P3(112) et P4(426).
  État initial : 100 500 200 300 600
  État final : 100 P2+83 P3+88 P1+88 P4+174
  Taux de fragmentation = (100 + 83 + 88 + 88 + 174) / (100 + 500 + 200 + 300 + 600) $\simeq$ 0.3
- Worst fit : P1(212), P2(417), P3(112) et P4(426).
  État initial : 100 500 200 300 600
  État final : 100 P2+83 200 P3+188 P1+388 Placement de P4 impossible.

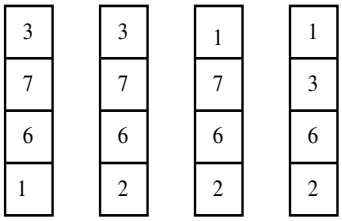
Quel algorithme utilise le meilleur ajustement sur cet exemple ?

La meilleure allocation ici est le Best Fit.

Exercice 4. Dérouler l'algorithme de remplacement de pages FIFO pour la suite de références suivantes, en supposant qu'on dispose de 4 blocs en mémoire centrale. Combien de défauts de page sont-ils produits ?

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6





Au total, 14 défauts de page.

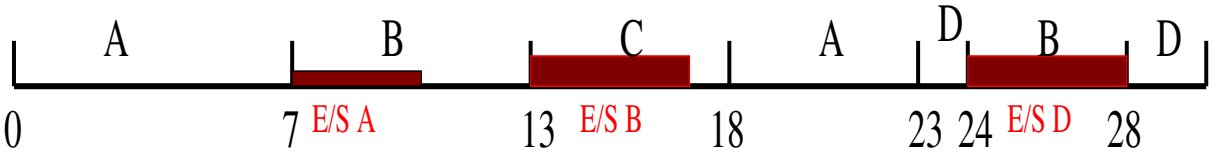
Exercice 5. On considère 4 processus A, B, C et D et on suppose que leur exécution est comme suit :

pour A : 7 unités de temps CPU, 3 unités de temps d'E/S et 5 unités de temps CPU.
pour B : 6 unités de temps CPU, 4 unités de temps d'E/S, 4 unités de temps CPU.
pour C : 5 unités de temps CPU.
pour D : 1 unité de temps CPU, 4 unités de temps d'E/S, 2 unités de temps CPU.

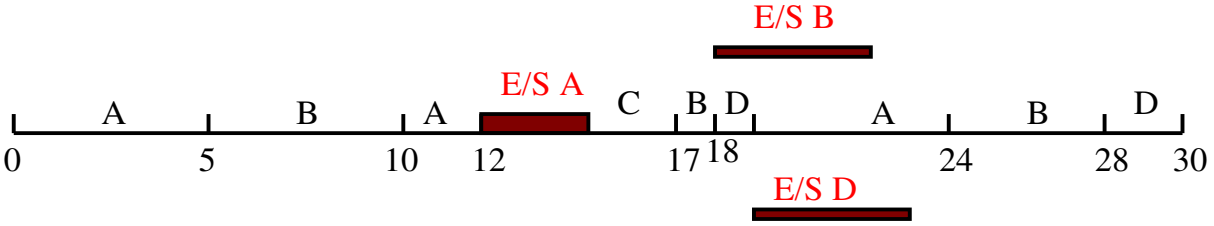
On suppose que A se présente à l'instant 0, B se présente à l'instant 1, C se présente à l'instant 9 et D se présente à l'instant 12.

Montrer comment les 4 processus utilisent le processeur dans chacun des cas suivants et indiquer quand les E/S de chaque processus s'exécutent :

1. chaque processus a son propre périphérique d'E/S et l'ordonnanceur fonctionne selon l'ordre FIFO (sans préemption).



2. chaque processus a son propre périphérique d'E/S et l'ordonnanceur utilise l'algorithme du tourniquet, avec un quantum de temps égal à 5. Le temps de commutation est nul.



3. Les trois processus A, B et D utilisent le même périphérique d'E/S et la file d'attente est gérée par le principe du premier arrivé, premier servi. L'ordonnanceur utilise l'algorithme du tourniquet, avec un quantum de temps égal à 5. Le temps de commutation est nul.