Licence d'Informatique

Système

Feuille de travaux dirigés n°6 – Correction

Synchronisation

1.

a. Les programmes associés aux processus P_1 et P_2 sont les suivants :

Processus $P_1 \equiv$
  répéter
    $<$ section restante 1 $>$;
    tant que *p2_dedans* faire rien ;
    p1_dedans := vrai ;
    $<$ section critique 1 $>$;
    p1_dedans := faux ;
  jusqu'à faux ;

Processus $P_2 \equiv$
  répéter
    $<$ section restante 2 $>$;
    tant que *p1_dedans* faire rien ;
    p2_dedans := vrai ;
    $<$ section critique 2 $>$;
    p2_dedans := faux ;
  jusqu'à faux ;

b. Supposons qu'il y ait blocage. Alors les deux processus sont bloqués sur leur boucle tant que ... faire rien respectives. D'où *p1_dedans* = *p2_dedans* = vrai. Mais cela implique que P_1 et P_2 sont tous deux en section critique ; contradiction.

c. On vient de montrer qu'il y a absence de blocage. Pour montrer que les programmes ci-dessus ne sont pas une solution au problème de la section critique, il faut montrer soit qu'il n'y a pas exclusion mutuelle, soit qu'il n'y a pas de progression, soit qu'il n'y a pas d'attente bornée.

Supposons les deux booléens à faux et les deux processus dans leur section restante (situation initiale). Considérons alors l'exécution suivante :

P_1 :	section d'entrée 1 : le test *p2_dedans* est faux
P_2 :	section d'entrée 2 : le test *p1_dedans* est faux
P_1 :	*p1_dedans* := vrai
P_2 :	*p2_dedans* := vrai
P_1 :	en section critique
P_2 :	en section critique également !

Il n'y a donc pas exclusion mutuelle, et ce n'est donc pas une solution au problème de la section critique.

2.

a. P_2 étant bloqué sur la boucle tant que on sait que D_2 vaut vrai. Si P_1 exécute sa section de sortie puis d'entrée, il va mettre D_1 à faux puis à vrai et $Tour$ à 2. Dès lors, il bloque sur sa boucle tant que car l'on a D_2 et $Tour = 2$. Il ne peut donc précéder P_2 en section critique. Dès que le CPU est donné à P_2, celui-ci peut entrer en section critique car D_1 et $Tour = 1$ n'est plus vrai ($Tour$ vaut maintenant 2).

b. **exclusion mutuelle :** On construit l'automate en suivant la méthode donnée dans le cours et l'on constate que les états où les deux processus sont en section critique ne sont pas atteignables.

absence de blocage : Supposons les deux processus bloqués. Alors ils sont tous deux sur leurs boucles `tant que` et l'on a donc

$$(D_1 \text{ et } Tour = 1) \quad \text{et} \quad (D_2 \text{ et } Tour = 2)$$

ce qui est impossible.

condition de progression : Supposons P_1 en section restante. Alors D_1 vaut faux. Dès lors, P_2 peut entrer en section critique car D_1 et $Tour = 1$ ne sera jamais vrai.

attente bornée : Démontrée dans la question *a*. En effet, on a vu qu'un processus en attente de section critique attendra l'exécution d'au plus une section critique de l'autre processus.

3. Dans cet exercice, on va utiliser l'invariant suivant :

À tout moment de l'exécution, il y a exactement une valeur vrai *parmi les* $n+1$ *valeurs booléennes* $libre, tour[1], \ldots, tour[n]$.

Cet invariant est clairement maintenu car les seules opérations modifiant ces valeurs booléennes sont les instructions *Swap*, supposées atomiques.

a. On remarque que seul le processus i peut modifier la valeur de $tour[i]$. Dès lors, si le processus i est en section critique, on sait que $tour[i]$ vaut vrai (car on est sorti de la boucle tant que et l'on n'a pas modifié $tour[i]$ depuis). Supposons deux processus i et j en section critique. Alors $tour[i]$ et $tour[j]$ valent vrai tous les deux, ce qui contredit l'invariant ci-dessus. On a donc bien l'exclusion mutuelle.

b. La réponse est oui. En effet, supposons que plusieurs processus soient en attente de section critique. Dès que le booléen *libre* reprend la valeur vrai, alors le premier de ces processus qui obtient le CPU « prendra » cette valeur vrai et entrera en section critique. Ce peut être n'importe lequel, et même toujours le même, selon la façon dont l'ordonnancement est réalisé.

c. Comme on le constate, les processus réalisent une *attente active* : ils passent en effet leur temps à exécuter des instructions *Swap*.