Université de Strasbourg
UFR Mathématique et Informatique

M1 Informatique
2020/2021

Algorithmes Distribués

TD n° 2 : Exclusion Mutuelle

I Algorithme de la boulangerie (Lamport 74)

On considère quatre processus P_0, P_1, P_2, P_3 qui exécutent cet algorithme pour accéder en section critique.
Q 1. Montrez pour le scénario ci-dessous :

1. Une évolution possible des variables $num[i]$ et $choix[i]$ pour tout i,
2. L'ordre dans lequel les différents processus accèdent à la section critique.
3. Y a-t-il différents ordres possibles d'accès des processus à la section critique ?
4. Quel est le rôle du tableau choix ?

Scénario :
t1 : P1 est entrant ;
t2 : P2 est entrant ;
t3 : P2 est dedans ;
t4 : P3 est entrant ;
t5 : P3 est dedans ;
t6 : P0 est entrant ;
t7 : P1 est dedans ;
t8 : P0 est dedans ;

Q 2. Preuve de l'algorithme :

1. Montrez que ce protocole est exempt d'interblocage, et qu'il assure l'équité des processus.
2. Montrez que exclusion mutuelle est assurée.

II Un jeton et un algorithme (Ricart/Agrawala)

Supposons plusieurs processus répartis sur des sites distants. Ces processus partagent une même ressource et cette dernière ne peut être utilisée que par un processus à la fois. Pour garantir le bon fonctionnement d'un tel système, il faut mettre en place un algorithme d'exclusion mutuelle.

Une section critique est une zone de code concernant la ressource partagée : un seul processus au plus doit être en section critique afin de garantir une utilisation correcte de la ressource. On distingue généralement trois approches pour gérer l'exclusion mutuelle dans un système réparti :
— approche centralisée (utilisation d'un coordinateur) ;
— approche par jeton ;
— approche par liste d'attente répartie.

Les protocoles d'exclusion mutuelle partagent les propriétés suivantes :
— **Définition** : à tout instant un processus au plus en section critique ;
— **Atteignabilité** : si plusieurs processus sont bloqués en attente de la section critique alors qu'aucun processus n'est en section critique, alors l'un d'eux doit y accéder en un temps fini ;
— **Progression** : un processus en attente accède à la section critique en un temps fini ;
— **Indépendance des parties conflictuelles et non-conflictuelles** : un processus hors de la section critique ou du protocole d'entrée ne doit pas influer sur le protocole d'exclusion mutuelle ;
— **Banalisation de la solution** : aucun processus ne joue de rôle privilégié.

Pour qu'une solution soit valide, il faut démontrer que ces propriétés sont bien respectées.
Soit 4 processus P_0, P_1, P_2, et P_3 qui se trouvent dans la situation suivante :

- P_3 est dans la section critique S
- P_1 demande d'entrer dans la section critique S à T_0 avec une horloge locale (estampille) de 1.
- P_0 demande d'entrer dans la section critique S à T_1 avec une horloge locale (estampille) de 5.
- P_2 demande d'entrer dans la section critique S à T_2 avec une horloge locale (estampille) de 3.
- P_3 sort de la section critique à T_3

On suppose que $T_0 < T_1 < T_2 < T_3$.

Algorithme : pour un processus P_i

```
Demande_Entree_SC()
{
    estampille ++;
    requete.emetteur = i;
    requete.date = estampille;
    demande[i] = estampille;
    diffuser(requete);

    si ( ! jeton_present) :
        attendre((jeton, new_tampon));

    tampon = new_tampon;
    dedansSC = VRAI;
    jeton_present = VRAI;
}
```

```
Reception_Requete()
{
    k = requete.emetteur;
    demande[k] = max(demande[k], requete.date);

    si (jeton_present) && ( ! dedansSC) :
        Sortie_SC()
    sinon rien (pas de jeton)
}

Sortie_SC()
{
    tampon[i] = estampille;
    dedansSC = FAUX;

    pour j=(i+1)% N; j==i; j=(j+1)% N :
        si (demande[j] > tampon [j]) && jeton_present) :
            jeton_present = FAUX;
            envoyer((jeton, tampon), j);
}
```

Q 1. Dessinez le chronogramme des différents événements.
Q 2. Preuve de l'algorithme :

1. Montrez que l'exclusion mutuelle est assurée ;
2. Montrez l'absence d'inter-blocage ;
3. Montrez que l'algorithme est équitable (les messages sont délivrés en un temps fini).

Q 3. Donnez le nombre de messages nécessaires pour une entrée en SC.
Q 4. Déroulez l'algorithme d'exclusion mutuelle basée sur un jeton pour déterminer les événements de chaque processus (valeur du jeton, section critique, etc.).

Evénements	0	1	2,3,4	5	6,7,8	9
Dedans	Faux					
Estampille	4					
Jeton	Faux					
Demande	(4,0,2,X+1)					
Tampon	(4,0,2,X)					
Action	/					
Dedans	Faux					
Estampille	0					
Jeton	Faux					
Demande	(4,0,2,X+1)					
Tampon	(4,0,2,X)					
Action	/					
Dedans	Faux					
Estampille	2					
Jeton	Faux					
Demande	(4,0,2,X+1)					
Tampon	(4,0,2,X)					
Action	/					
Dedans	Vrai					
Estampille	X+1					
Jeton	Vrai					
Demande	(4,0,2,X+1)					
Tampon	(4,0,2,X)					
Action	SC					

Evénements	10,11,12	13, 14	15,16	17,18	19
Dedans					
Estampille					
Jeton					
Demande					
Tampon					
Action					
Dedans					
Estampille					
Jeton					
Demande					
Tampon					
Action					
Dedans					
Estampille					
Jeton					
Demande					
Tampon					
Action					
Dedans					
Estampille					
Jeton					
Demande					
Tampon					
Action					

III Liste d'attente répartie (Lamport)

Avec une exclusion mutuelle basée sur une liste d'attente répartie, chaque processus gère une copie de la file d'attente. Cette dernière est mise à jour par des messages de requêtes et de libération de la section critique.

Chaque processus doit :
- — recevoir tous les messages de requêtes et de libération de tous les autres processus ;
- — savoir ordonner ces messages en fonction de leur date ⇒ utilisation d'horloges logiques.

Algorithme : pour un processus P_i

```
Init_Site()
{
    ∀ j F_i[j] = (Hors, 0, J);
    H_i = 0;
}

Entree_SC()
{
    Diffuser(requete, H_i, i);
    F_i[i] = (requete, H_i, i);
    H_i ++;
    Attendre que ∀ j ≠ i :
        estampille de F_i[i] < estampille F_i[j];
}

Sortie_SC()
{
    Diffuser(Hors_SC, H_i, i);
    F_i[i] = (Hors_SC, H_i, i);
    H_i ++;
}
```

```
Reception_message((type, H, j))
{
    H_i = max(H,H_i) + 1;

    switch(type) :
    {
        case requete :
            F_i[j] = (requette,H,j);
            if type(F_i[i] ≠ requete) :
                envoi(Ack, H_i, i) → P_j;

        case Hors_SC :
            F_i[j] = (Hors_SC, H, j);

        case Ack :
            F_i[j] = (Ack, H, j);
}
```

Q 1. Reprenez le scénario précédent en l'ajustant pour dérouler l'algorithme d'exclusion mutuelle basée sur une liste d'attente répartie afin de déterminer les événements de chaque processus (entrée en section critique, etc.). Petite différence, P_2 demandera à entrer en section critique à T_2 avec une horloge locale de 6.

Q 2. Preuve de l'algorithme :

1. Montrez que les demandes d'entrée en section critique sont traitées selon l'ordre total imposé par le mécanisme d'estampillage ;
2. Montrez que le protocole assure l'exclusion mutuelle.

Q 3. Quel est le nombre de messages émis pour mettre en œuvre un accès à la section critique ?

Evénements	0	1	2	3	4	5	6	7
H_0	4							
$F_0[0]$	hs,0,0							
$F_0[1]$	hs,0,1							
$F_0[2]$	hs,0,2							
$F_0[3]$	rq,0,3							
Action	/							
H_1	1							
$F_1[0]$	hs,0,0							
$F_1[1]$	hs,0,1							
$F_1[2]$	hs,0,2							
$F_1[3]$	rq,0,3							
Action	/							
H_2	1							
$F_2[0]$	hs,0,0							
$F_2[1]$	hs,0,1							
$F_2[2]$	hs,0,2							
$F_2[3]$	rq,0,3							
Action	/							
H_3	1							
$F_3[0]$	ack							
$F_3[1]$	ack							
$F_3[2]$	ack							
$F_3[3]$	rq,0,3							
Action	SC							

Evénements	8	9	10	11	12	13	14	15	16
H_0									
$F_0[0]$									
$F_0[1]$									
$F_0[2]$									
$F_0[3]$									
Action									
H_1									
$F_1[0]$									
$F_1[1]$									
$F_1[2]$									
$F_1[3]$									
Action									
H_2									
$F_2[0]$									
$F_2[1]$									
$F_2[2]$									
$F_2[3]$									
Action									
H_3									
$F_3[0]$									
$F_3[1]$									
$F_3[2]$									
$F_3[3]$									
Action									