Université de Strasbourg **Licence 3 Informatique**
UFR Mathématique et Informatique **Année 2014/2015**

Systèmes Distribués

Contrôle Continu - Épreuve finale (50% de l'UE)

8 avril 2015

Durée : 2h00
Aucun document autorisé.

Partie I : Questions de cours (8 points)

Justifiez vos réponses en utilisant des exemples si besoin (1 point par question).

a- Donnez des exemples concrets de systèmes et d'algorithmes distribués (deux de chaque).
Systèmes :
NFS/Samba -> système de fichiers distribué
DNS -> service de noms hiérarchique
GIT/SVN -> système de gestion centralisé
Algorithmes :
Bellman Ford distribué -> calcul des plus courts chemins dans RIP
CBCAST/ABCAST -> diffusion atomique
Algorithme de détection d'Inter-blocage -> détecter la présence d'un cycle

b- Comment et pourquoi déterminer un ordre entre les événements ayant lieu sur différent sites ?
Pour assurer la cohérence (forte ou causale) des données réparties sur plusieurs sites, la reprise sur panne d'un gros calcul réparti, ou encore l'exclusion mutuelle... Pour y parvenir on peut utiliser des horloges scalaires, vectorielles ou matricielles qui enregistrent +/- finement l'activité des autres sites.

c- Comment assurer une diffusion atomique en fonction du contexte (quelles sont les hypothèses et les méthodes associées).
atomique := uniformité ABCAST ou garantie sur l'ordre causale CBCAST.
ABCAST et CBAST ne font pas d'hypothèse spécifiques sur la nature du graphe de communications (p.e non FIFO par exemple). On peut simplifier ces algos si on dispose de certaines garanties.

d- Quels sont les défauts des méthodes d'inter-blocages par prévention ?
Comment définir les priorités (niveau ressource ou processus) ? Trop arbitraire en pratique... Et surtout, ces techniques (deux approches : attendre ou mourrir OU attendre ou tuer) peuvent ralentir l'exécution du système voire l'utiliser inutilement (si les procs sont tués pour rien).

e- Quelle information doit circuler dans les messages de détection d'inter-blocages ? À qui envoyer ces messages ?
La liste des sites impliqués dans une chaine de dépendance pour détecter l'occurence d'un cycle. Ces messages doivent être transmis aux sites avec lesquelles il existe une dépendance inter-sites.

f- Quel est l'algorithme permettant de définir un ordonnancement minimal ? Spécifiez le en quelques lignes de pseudo-code. L'algorithme de Bellman permet de définir les dates au plus tôt : sur cette base, on a tout ce qu'il faut pour définir un ordonnacement minimal si le nombre de processeurs n'est pas limité. On marque un noeud si tous ces preds sont déjà marqués et on lui affecte, en tant que date au plus tôt, le max de leurs dates au plus tôt + leurs durées d'execution. A l'initialisation seule la tache virtuelle de départ est marqué avec une date au plus tôt de 0.

g- Quels sont les points communs et les différences entre une approche de chiffrement asymétrique (e.g. RSA) et celle de Diffie-Hellmann ?
Point commun : on peut faire circuler l'info nécessaire au chiffrement sur un canal non sûr (pas de problème avec attaque MIM passif).
Point divergent : syémtrique pour Diffie Helmman, asymétrique pour RSA.

h- Quels sont, selon vous, les avantages et inconvénients des différentes approches de structuration logique de niveau réseau (i.e. le graphe de communication) ?
Trois options :
– étoile : centralisé donc sensible à tous les pbs liés au serveur : panne ? goulet d'étranglement système et réseau ! mais facile à mettre en oeuvre.
– anneau : lent si peu exploité mais efficace (et déterministe/prévisible) si bcp de charge et presque aussi facile que centralisé.
– maillage quelconque : robuste, élégant (car extensible) mais plus difficile à mettre en oeuvre.

Partie II : Inter-blocages (6 points)

Dans le système sur lequel nous allons travailler il existe 4 processus (P_i, i=1..4) qui partagent 6 types de ressources (R_j, j=1..6). Le système possède une seule ressource de chaque type. L'état courant du système est décrit par les matrices d'allocation (cf. tableau 1) et de besoin (cf. tableau 2) suivantes, ainsi que le vecteur de disponibilité instantanée.

	R1	R2	R3	R4	R5	R6
P1	1	0	1	0	0	0
P2	0	0	0	1	0	0
P3	0	0	0	0	1	0
P4	0	0	0	0	0	1

TABLE 1 – Ressources allouées.

	R1	R2	R3	R4	R5	R6
P1	0	1	0	0	0	0
P2	0	1	1	0	0	0
P3	1	1	0	0	0	0
P4	0	1	1	1	0	0

TABLE 2 – Ressources demandées.

a- L'état courant est-il sûr par rapport aux éléments dont vous disposez ? Pourquoi ?
Oui, car, selon les éléments fournis (et avec des hypothèses favorables), il existe un scénario possible pour satisfaire tous les processus : ici, le vecteur de disponibilité permet de satisfaire P1 puis P2, etc, jusqu'au dernier processus.

b- Si P3 demande la ressource R2, peut-on satisfaire immédiatement sa requête ? Justifiez et expliquez les effets produits pour continuer la progression.
Non dans un scénario défavorable. Si P3 obtient R2 sans le relâcher alors qu'il attend toujours R1, on peut conclure à la création d'un interblocage entre les procs P3 et P1 avec R1 et R2 comme ressource (qui a déjà R1 et R3 et n'attend plus que R2 pour terminer).

c- Si P1 demande la ressource R2, peut-on satisfaire immédiatement sa requête ? Justifiez et expliquez les effets produits pour continuer la progression.
Oui, parce que P1 va utiliser R2 et on peut supposer ensuite qu'il va se terminer, libérant ainsi R1, R2 et R3. P2 pourra débuter puis se terminer en libérant R2-4 (R1 étant toujours dispo). P3 peut alors utiliser R1 et R2 en plus de R5 dont il dispose déjà. Une fois P3 terminé, il libère ces 3 ressources et permet à P4 de s'exécuter. Il s'agit donc d'un état sûr, on peut donner R2 à P1 directement.

d- Selon vous, quelles sont les limites de l'algorithme du banquier pour une utilisation dans une situation réelle ? Justifiez.

- Coûteux : L'algorithme est en effet très coûteux en temps d'exécution et en mémoire pour le système puisqu'il faut maintenir plusieurs matrices, et déclencher à chaque demande de ressource, l'algorithme de vérification de l'état sain qui demande $m \times n$ opérations. (m est le nombre de types de ressources et n est le nombre de processus).
- Théorique : l'algorithme exige que chaque processus déclare à l'avance les ressources qu'il doit utiliser, en type et en nombre. Cette contrainte est difficile à réaliser dans la pratique.
- Pessimiste : l'algorithme peut retarder (voire empêcher) une demande de ressources dès qu'il y a risque d'interblocage (mais en réalité l'interblocage peut ne pas se produire).

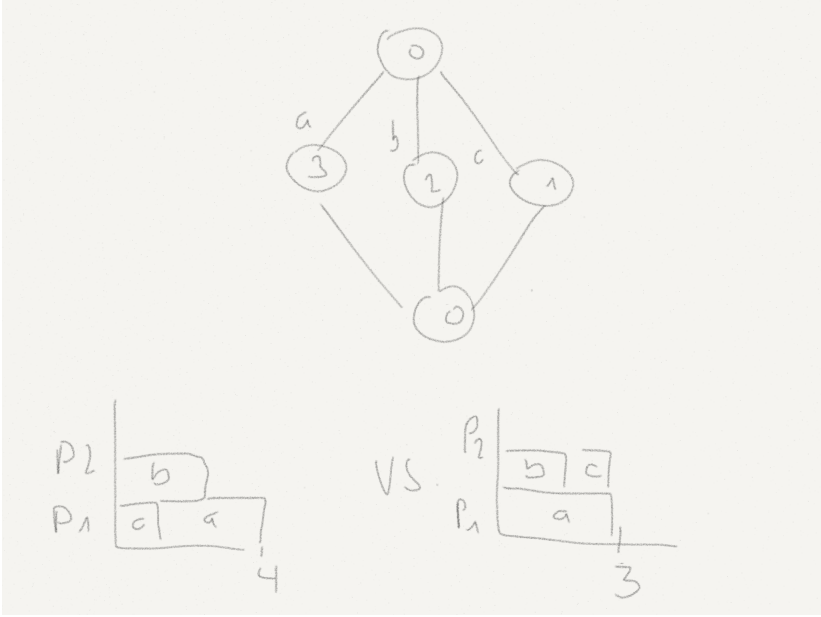
e- Construisez, sur base de l'exemple fourni ici, un cas où l'algorithme d'évitement du banquier échoue malgré l'existence d'une solution viable.
La question b- est un bon exemple : le banquier dit non car il a peur que P3 ne libère pas R2 dans la prochaine tâche (en effet il ne connait pas le détail inter-tâches, il fait au pire sur l'ensemble) alors qu'il est possible de libérer simplement cette ressource avant la prochaine tâche.

Partie III : Ordonnancement (6 points)

Dans cette dernière partie, nous allons considérer l'utilisation de l'algorithme de Coffman & Graham selon différents contextes.

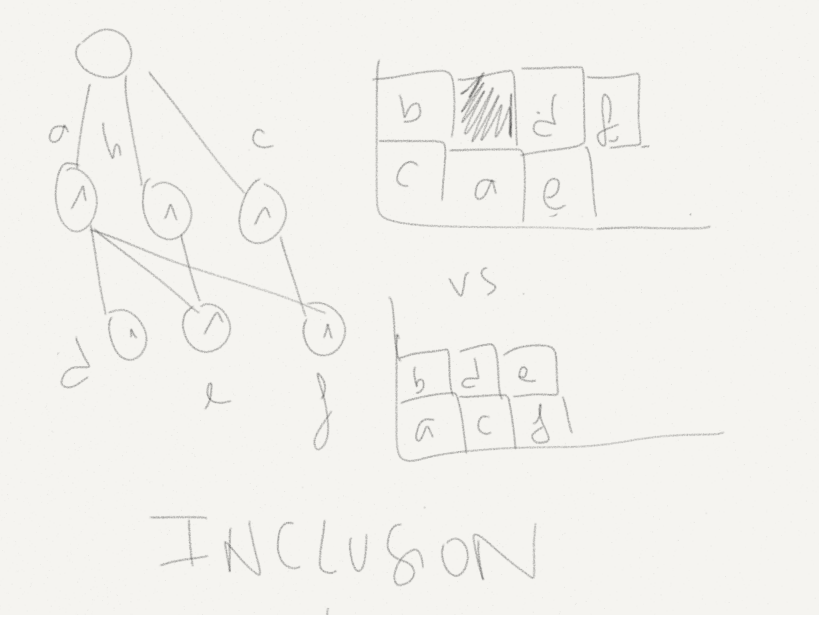
a- Dans quelles configurations (i.e. selon quelles hypothèses en particulier) cet algorithme est optimal ? Et minimal ?
Cet algorithme est optimal (avec l'aide préalable des dates au plus tard) dans le cas où p=2 procs. Toutes les autres hypothèses du cours doivent être valables (pas de couts de communication, sous taches indivisibles, etc). L'algorithme ne permet pas de garantir la minimalité (peu importe les hypothèses dès lors que le nb de procs est limité). La solution est minimale uniquement si, par chance, le nb de procs à disposition est suffisant.

b- Montrez sur un petit exemple avec $p = 2$ processeurs que les dates au plus tard doivent être prises en compte pour réaliser l'ordonnancement optimal.



Sans les dates au plus tard (0,1,2), l'ordonnancement ne serait pas optimal sur deux processeurs.

c- Montrez sur une variante de l'exemple précédent (mais toujours avec $p = 2$ processeurs) l'interêt de la principale règle d'étiquetage mise en oeuvre par cet algorithme.
La seule règle d'étiquetage utile est l'inclusion de la liste des successeurs grace à la prise en compte de l'ordre lexicographique.



d- Construisez un graphe de dépendance le plus simple possible où cet algorithme échoue à trouver une solution optimale pour $p = 3$ processeurs.
Pénible dans le temps imparti !