M2 COMASIC – Examen
Sûreté de Fonctionnement
28/01/2019

Durée de l'examen 2h.

Consignes particulières : les documents sont interdits. Les téléphones portables doivent être rangés. Le présent examen est noté sur 20 pts. Le barème est indiqué devant chaque question à titre indicatif.

I Principes de la sûreté de fonctionnement (3 pts)

Question 1) (1 pt) Donnez un exemple d'exigence non-fonctionnelle de sûreté de fonctionnement

Rappel : une exigence est non fonctionnelle si elle ne repose pas directement sur la définition d'un comportement attendu (par exemple si on ne dit pas « Lors qu'une défaillance temporelle est détectée, le système ne doit plus jamais produire de résultat ». Une exigencce non fonctionnelle repose sur un attribut (souvent numérique ». Ainsi, une exigence non fonctionnelle pourrait être : »La probabilité que le système soit à la date t dans un état disponible est 99 % »

Question 2) (1 pt) Définissez l'élimination des fautes et la tolérance aux fautes, expliquez en quoi ces deux pratiques sont complémentaires.

L'élimination de faute consiste à identifier les fautes (l'élément qui permet leur activation et à le retirer du système). Par exemple, un pointeur non initialisé est une faute de développement pouvant entraîner l'arrêt d'un programme, initialiser le pointeur ou ne plus utiliser le pointeur élimine la faute

Question 3) (1 pt) Définissez brièvement ce qu'est une zone de confinement d'erreur dans une description architecturale d'un système.

Une zone de confinement d'erreur est une unité de description d'une architecture tolérante aux fautes telle qu'elle peut tolérer certaine faute d'interaction, et/ou certaine faute activée en interne en empêchant la propagation des erreurs sur son interface. En complément, la zone doit signaler son état de fonctionnement dans le cas où elle ne serait pas capable de confiner les erreurs en interne.

II Architecture redondantes et modèles de fautes (8 pts)

Question 4) Supposons que l'on dispose d'une bibliothèque codée dans le langage de votre choix (C ou Java). Dans cette bibliothèque, on suppose que l'on dispose d'une

fonction/méthode implémentant le calcul de la racine carrée d'un nombre à virgule flottante, le paramètre d'entrée a pour type double, le type de la valeur renvoyée est aussi double.

a) (1 pt) Expliquez le principe du mécanisme de l'enveloppe de fonction en général pour une fonction pouvant défaillir soit à cause de fautes activées lors de l'exécution du code, soit à cause de fautes d'interaction.

Le principe de l'enveloppe est de développer une fonction qui dont le rôle est d'implémenter une zone de confinement au niveau d'une fonction. Ceci permet notamment d'insérer des tests de vraisemblance sur les valeurs de sortie de la fonction, ou sur les valeurs de ses paramètres d'entrée.

b) (1 pt) Illustrez ce principe pour la fonction racine carrée en précisant en quoi pourrait consister l'enveloppe (si l'on suppose que les fautes activées localement engendre des défaillances entraînant la production d'un résultat négatif).

```
Env_sqrt ( float f) {

if (f < 0) {

return -1}

else { float res=sqrt(f) ;

        if (res<0) || abs (res*res -f)/f >0.1{

return -1}

        else return res ;

}
```

// notation : tout version ou vous contrôler le résultat de sortie => 1pt si oubli mais contrôle valeur d'entrée 0.5.

Question 5) Nous supposons que l'on souhaite améliorer la fiabilité d'une fonction de calcul dont le temps d'exécution est inférieur à T si l'exécution est correcte.

a) (1,5 pt) Expliquer le principe de fonctionnement de la stratégie de réplication active pour N répliques. (architecture, comportement et hypothèse(s) permettant de justifier l'intérêt de l'architecture)

La réplication active consiste à exécuter en parallèle N versions d'une même fonction de telle sorte que l'on puisse comparer les résultats produit pour détecter et masquer un certains nombre de défaillances. Le comportement se déroule en 3 phase, d'abord on déclenche les calculs sur les N répliques (en envoyant les données d'entrée par exemple), puis le calcul est réalisé et chaque réplique transfert à un composant appelé voteur son résultat. Le voteur collecte les résultats pendant un certain temps (borné) puis transmet la valeur la plus reçue comme résultat. Cette architecture fonctionne si le nombre de version fonctionnant correctement est supérieure strictement à la moitié des répliques déployées

b) (0,5 pt) En faisant l'hypothèse que toute faute s'activant dans une réplique déclenche une défaillance byzantine, indiquer combien d'activations de fautes peuvent être tolérées dans le pire scénario pour une architecture à 9 répliques (dans le cas où les défaillances sont permanentes et où il n'y a pas de réparation).

4

Question 6) Supposons maintenant que chaque réplique d'une architecture de réplication active à 5 répliques possède 2 modes de défaillance différents : crash et byzantin. La différence est que même dans le pire cas, une défaillance par crash se traduit par l'absence de résultat. On souhaite modifier l'architecture pour s'adapter à l'hypothèse suivante : « le nombre de défaillance byzantine est toujours inférieur au nombre de répliques fonctionnant correctement ». On notera en revanche que le nombre de défaillances par crash n'est pas contraint.

a) (0,5) Sous l'hypothèse énoncée ci-dessus quel(s) est(sont) les mode(s) de défaillance(s) possible(s) de l'architecture complète (à 5 répliques)?

Le crash uniquement

b) (1,5 pts) Le composant en charge de produire le résultat final d'une telle architecture est appelé souvent « voteur », son comportement est d'attendre N valeurs puis de produire la valeur majoritaire. Expliquez comment il faudrait modifier / adapter ce comportement pour 1) ne pas attendre indéfiniment les répliques crashées, 2) pouvoir produire une réponse le plus tôt possible sans avoir à attendre la fin d'exécution de répliques défaillantes byzantines.

Le voteur sera changé pour un composant qui tient le décompte des répliques « crashée » et mémorise leur identité (Soit C le nombre de processus marqués crashés). Pour faire cela, il collecte les résultat jusqu'à la date T et tout réplique n'ayant pas fourni de résultat est marquée crashée. De plus, dans le cas, où P résultats identiques on déjà été reçus P > (5-C)/2, il produit la valeur reçu P fois sans attendre les résultats suivants.

Ainsi le choix du résultat majoritaire ne se fait plus que sur un sous ensemble des réplique qui ne contient plus que des répliques soit correctes soit byzantine. De plus, ce choix se fait au plus tôt si suffisamment de processus correct ont transmis leur résultat. (Notez que ceci reste vrai même si l'estimation courante du nombre de processus crashés est fausse car sous estimée grâce à l'hypothèse en début de q6).

c) (2 pts) Identifiez l'ensemble des combinaisons de défaillances qui n'empêche pas l'architecture de produire une réponse correcte (si l'on suppose le comportement décrit au début de la question 6 et le voteur décrit en 6b))

les scénarios valides satisfont f< (5-C)/2

III Mécanismes de tolérance aux fautes (5 pts)

Question 7) (1 pt) Rappelez le principe du mécanisme de recouvrement avant.

Le recouvrement avant consiste sur détection d'erreur à écraser l'état courant par une valeur par défaut dite sûre.

Optionnel : cette valeur sûre permet en règle générale d'éviter une défaillance dont on ne maîtrise pas les conséquence. C'est ce que font la plupart des appels systèmes.

Question 8) (2 pts) Indiquez en justifiant si la recouvrement arrière peut être appliqué de manière raisonnable pour une application en fonction des fautes suivantes (on supposera que la détection est parfaite … i.e. elle a lieu dès activation)

1. Pointeur (adresse de variable) non initialisé à la déclaration mais utilisé la ligne suivante (déréférencé).

2. Connexion réseau temporairement perdue entraînant la défaillance d'un appel système cherchant à exploiter les communications
3. Interférences physiques temporaires avec le matériel affectant uniquement les données de l'application
4. Interférences physiques temporaires avec le matériel affectant le code et les données de l'application.

1. non si cette partie du code est toujours exécutée, oui sinon
2 : oui le recouvrement arrière permet de « retenter » l'appel système avec une chance de succès
3 oui si la sauvegarde est elle même protégée ou que l'interface a une faible chance de toucher toute la mémoire, non sinon
4. non aucune chance

Question 9) (2 pts) Tolérance aux fautes pour les données et entrelacement
On suppose que l'on dispose d'une manière de stocker une information codée sur 45 bits sur des blocs de 64 bits de telle sorte que l'on sait tolérer 9 bits corrompus (car la distance de hamming sur ce code est de 19=2*9+1). **En supposant que l'on a à stocker N blocks en mémoire mais qu'une faute peut altérer jusqu'à 4 octets contigus, proposer une manière d'utiliser le codage par bloc décrit précédemment pour tolérer ces « bursts » de bits erronés en le couplant à une stratégie d'entrelacement du contenu stocké (i.e. décrivez votre solution sur le nombre de blocs suffisant pour l'illustrer).**

L'idée est de considérer au moins 4 blocs de 64 bits M1,M2 M3 M4, chacun divisés en 8 sous blocks : M1 → $M1_1$,$M1_2$,$M1_3$,$M1_4$,$M1_5$,$M1_6$,$M1_7$,$M1_8$.
Ensuite, on stocke en mémoire $M1_1$ $M2_1$,$M3_1$,$M4_1$,$M1_2$,$M2_2$,$M3_2$,$M4_2$ $M1_3$…
Cette manière de faire l'entrelacement garanti que sur 4 octets consécutifs en mémoire seul 1 correspond à 1 block donné. Cela signifie donc qu'au plus 8 bits auront été altéré ce qui est tolérable.

IV Raisonnement autour de l'analyse quantitative (4 pts)

Il est dit qu'une architecture de réplication passive permet d'augmenter la disponibilité d'une fonction de calcul si les défaillances correspondent à des défaillances par crash. Nous allons supposer que l'on dispose de 4 répliques et que le crash est du à une défaillance matérielle qui se produit indépendamment des calculs réalisés :que le système soit la réplique primaire (celle qui réalise les calcul) ou une réplique secondaire (back-up stockant la dernière sauvegarde). Ainsi, une réplique est soumise au même conditions d'occurrence des défaillances.

Question 9) (1 pt) Supposons que la probabilité pour une réplique de défaillir avant 100h de fonctionnement est de 10^{-2}. Que dire de la probabilité que l'architecture répliquée complète ait défailli avant 100h de fonctionnement ? (on supposera que les événements de défaillance sont des événements aléatoire indépendants).

Il faut que toutes les répliques ait défailli = $p^4 = 10^{-8}$.

Question 10) (0,5 pts) Nous souhaitons maintenant modéliser le fait que plus le temps de traitement dure plus la probabilité de crash augmente : quel modèle est le plus adapté entre une chaîne de Markov à temps continu et une chaîne de Markov à temps discret ?

Chaine à temps continu …

Question 11) (2,5 pts) Proposez un modèle à base d'une ou plusieurs chaînes de Markov permettant

a) de capturer le fait que plus le temps s'écoule, plus la probabilité de défaillance augmente (indépendamment de l'application exécutée).

b) de tenir compte de l'indépendance de l'occurrence des défaillances de répliques

c) de tenir compte du fait qu'en moyenne une réplique défaille au bout de T unité de temps.

d) de capturer pour chaque réplique le fait que la détection du crash prend en moyenne T_{det} unités de temps (indépendamment sur chaque réplique) et qu'après cette détection la réparation du crash prend en moyenne T_{rep} unités de temps.

La définition d'une telle chaîne devra contenir une description du sens pratique de ses différents états.

On peut considérer 4 chaînes identiques à 3 état : fonctionnel (état 1) crashé non détecté (0), crashée détectée(-1).

(1)-→ (0) taux 1/T

(0) → (-1) taux 1/Tdet

(-1) → (1) taux 1/Trep

Solution alternative : comptage des crashés nd / crashé d/ fonctionnels

(4,0,0) → (3,1,0) taux 4/T

(3,1,0) → (3,0,1) taux 1/Tdet

(3,1,0) → (2,2,0) taux 3/T

(3,0,1) → (2,1,1) taux 3/T

(3,0,1) → (4,0,0) taux 1/Trep

(2,1,1) → (1,2,1) taux 2/T

(2,1,1) → (2,0,2) taux 1/Tdet

(2,1,1) → (3,1,0) taux 1/Trep

(2,0,2) → (1,1,2) taux 2/T

(2,0,2) → (3,0,1) taux 1/Trep

(2,2,0) → (1,3,0) taux 2/T

(2,2,0) → (2,1,1) taux 2/Tdet

….vous l'aurez compris cette solution était super pénible mais facilement énumérable ….