SYSTÈMES TEMPS RÉEL EMBARQUÉS CRITIQUES

MASTER SAR / SE301 / UE Systèmes Temps Réel Embarqués Critiques

Barème indicatif, **sans document**
Les téléphones portables doivent être éteints et rangés dans vos sacs.

Il sera tenu compte de la présentation et de la clarté dans la rédaction.
Seules les réponses précises et justifiées seront considérées.

1 Analyse de Cache (3 points)

1.1 Analyse Must (1.5 point)

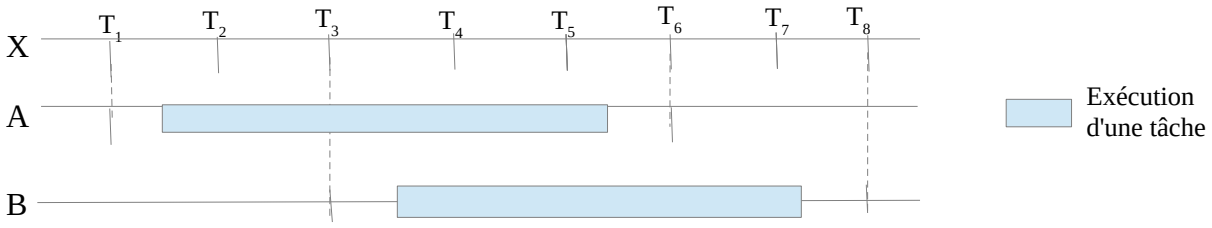
L'analyse **Must** est basée sur l'interprétation abstraite et trace des informations concernant des bloc de mémoire (memory block). Quelle type d'information est tracé par cette analyse (domain) pour un cache associatif avec 4 ensembles (4-way set-associative) ? Expliquez la signification de ces informations. Expliquez les opérations qui manipulent ces informations durant l'analyse (fonctions de transfert et le opération join). Vous n'êtes pas obligé d'indiquer les formules.

1.2 Classification des accès (1.5 point)

En utilisant les informations calculées par des analyses du cache les accès aux blocs de mémoire (memory blocks) peuvent être classés. Nommez les **4** types de classification qui ont été traité en cours. Quel est le lien avec l'analyses de persistance ainsi que les analyses **May** et **Must** ?

2 Time-Triggered (3 points)

Un système est constitué de deux tâches A et B dont un extrait du comportement temporel est représenté sur le schéma ci-dessous.



La tâche A produit une variable temporelle X qui est consommée par la tâche B. Les dates de production d'une nouvelle valeur pour la variable temporelle X sont représentés sur le schéma (T_1 à T_8). Par ailleurs, les dates T_1 et T_3 correspondent respectivement à l'activation des tâches A et B. Les dates T_6 et T_8 correspondent respectivement aux échéances des ces tâches A et B activées aux dates T_1 et T_2.

Supposons que l'exécution du code de la tâche A aboutisse, entre les dates T_4 et T_5, à une écriture vers la variable temporelle X. Supposons également que l'exécution du code de la tâche B engendre, entre les dates T_6 et T_7, une lecture de la variable temporelle X.

2.1 Visibilité des valeurs (1,5 point)

Quelle est la date de visibilité associée à la valeur de la variable temporelle X lue par le code de la tâche B ? Quelle sera la date de visibilité associée à la valeur de la variable temporelle X écrite par le code de la tâche A ?

2.2 Règles de communication (1,5 point)

Formulez les règles de communication inter-tâche du modèle Time-Triggered?

3 Timing Failures (7 points)

Le modèle AADL ci-dessous fournit la description d'architecture logicielle d'un système temps réel implémenté en RT-POSIX.

```
data position
end position;

thread periodic
properties
  Dispatch_Protocol => Periodic;
end periodic;

thread periodic_pos extends periodic
features
  pos: requires data access position;
end periodic_pos;

thread implementation periodic_pos.t50
  annex behavior_specification {**
    states
      s1: initial complete final state;
    transitions
      s1 -[on dispatch]-> s1 {
                        computation(5 ms);
                        pos !<;
                        computation(5 ms);
                        pos !>
                      };
  **};
end periodic_pos.t50;
```

```
thread implementation periodic.t300
annex behavior_specification {**
states
   s1: initial complete final state;
transitions
 s1 -[on dispatch]-> s1 {computation(50ms)};
**};
end periodic.t300;

thread implementation periodic_pos.t100
  annex behavior_specification {**
    states
      s1: initial complete final state;
    transitions
      s1 -[on dispatch]-> s1 {
                        computation(30 ms);
                        pos !<;
                        computation(20 ms);
                        pos !> ;
                        computation(10 ms)
      };
  **};
end periodic_pos.t100;
```

```
system root_system
end root_system;

system implementation root_system.impl
subcomponents
  the_cpu: processor cpu;
  dis:     process   display.impl;
properties
    Actual_Processor_Binding => ( reference(the_cpu)) applies to dis;
    Scheduling_Protocol => (RMS) applies to the_cpu;
end root_system.impl;

process display
end display;

process implementation display.impl
subcomponents
  t_50  : thread periodic_pos.t50 {Period => 50 ms;};
  t_100 : thread periodic_pos.t100 {Period => 100 ms;};
  t_300 : thread periodic.t300 {Period => 300 ms;};
  pos   : data position {Concurrency_Control_Protocol => (PCP);};
connections
  cnx1: data access t_100.pos -> pos;
  cnx2: data access t_50.pos -> pos;
end display.impl;

processor cpu
properties
 Scheduling_Protocol => (POSIX_HIGHEST_PRIORITY_FIRST) ;
end cpu;
```

3.1 Politique d'ordonnancement (1 point)

Quel politique d'ordonnancement est utilisée dans ce modèle pour les tâches `t_50, t_100, et t_300` ? `POSIX_HIGHEST_PRIORITY_FIRST` ou `RMS` ? Justifiez votre réponse.

3.2 Ordonnançabilité (2 point)

Le système considéré est-il ordonnançable ? Justifiez votre réponse.
Indication : on considère que le protocole PCP promeut immédiatement un thread qui prend le verrou à la priorité maximum de l'ensemble des threads qui utilise le verrou. Lorsque le verrou est rendu, le thread correspondant reprend sa priorité d'origine.

3.3 Dépassement d'échéance (3 point)

Un collègue vous signale qu'il a constaté certains dépassements d'échéances à l'exécution. Ces dépassements d'échéances sont rares, mais posent de sérieux problèmes lorsqu'ils arrivent. Votre collègue vous demande de mettre en œuvre un mécanisme de détection de dépassement d'échéance. Décrivez l'architecture logicielle correspondante, en langage naturelle et/ou en pseudo-code AADL. Nous rappelons ici que le système s'appuie sur les services RT-POSIX.

3.4 Dépassement d'échéance II (1 point)

On considère que l'analyse d'ordonnancement conclue que le système est ordonnançable. Suggérez une amélioration du mécanisme de détection qui permettrait de détecter un risque de dépassement d'échéance avant la date de l'échéance proprement dite. Justifiez votre réponse.

4 Tolérance aux fautes (7 points)

Un ingénieur doit améliorer la fiabilité d'une fonction implémentée sous forme logicielle qui réalise un calcul servant à définir la vitesse de rotation d'un moteur à explosion. La fonction doit déterminer à intervalle fixe (une période) la rotation à appliquer à une pièce mobile dans le moteur. Ce logiciel est déployé sur des cartes embarquées interconnectées à un réseau pour constituer une architecture tolérante aux fautes par redondance. Le problème consiste à étudier différentes conceptions. Les fautes dont on redoute l'activation seront considérées dans un premier temps comme étant liées au matériel. Des erreurs arithmétiques et de contrôle d'exécution peuvent donc apparaître en raison des fortes vibrations et des perturbations électromagnétiques altérant l'état du processeur de manière aléatoire.

4.1 Mode de défaillance (1 point)

Expliquez la différence entre le mode de défaillance silencieux (appelé aussi crash), et le mode de défaillance byzantin ?

4.2 Réplication passive (2 point)

Rappelez le principe de fonctionnement de la réplication passive (lorsqu'il n'y a aucune défaillance et en présence de défaillance de type crash).

4.3 Stratégie de réplication (1 point)

Indiquez quelle stratégie de réplication serait la plus adaptée pour un tel système parmi celles que vous connaissez (passive ou active). Justifiez votre réponse et indiquez le nombre de répliques permettant de tolérer la défaillance de deux répliques logicielles sur une même période. (on supposera que chaque calculateur active ses fautes indépendamment)

Nous supposons à partir de maintenant que les défaillances du logiciel sont causées par son code et non le matériel. On suppose pour ces défaillances que le seul moyen de détection possible est la comparaison à la valeur retournée par une réplique non défaillante.

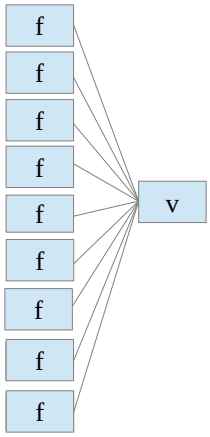
Le matériel peut toujours défaillir mais simplement en s'arrêtant de fonctionner. Lorsque l'on considérera N répliques du logiciel, on supposera que chaque réplique a un code différent (ainsi sur une même donnée, chaque réplique peut défaillir indépendamment des autres). De plus, ce n'est pas la défaillance d'une réplique logicielle qui cause celle du matériel.

Le vote sera aussi implémenté au niveau logiciel mais l'on supposera que ce code n'a pas de faute de programmation pouvant entraîner sa défaillance. Sur chaque calculateur, une système d'exploitation ARINC est déployé afin d'exécuter plusieurs répliques (ou des voteurs) sur un même calculateur.

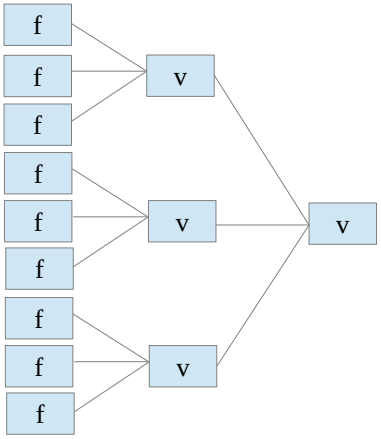
On supposera que d'un point de vue dimensionnement le coût d'un voteur est nul, et le coût d'une réplique logicielle est telle que seulement 2 répliques peuvent être déployées par calculateur.

4.4 Répliques logicielles (1 point)

Sur une architecture à 9 répliques logicielles. Justifiez pourquoi l'architecture de vote A1 est plus fiable que A2 (cf figure ci-dessous).



A1



A2

On suppose à partir de maintenant que l'on utilise un voteur déterminant la réponse majoritaire sur chaque calculateur. Chaque voteur reçoit les réponses des 9 répliques et produit son résultat (notez que les résultats produits sont identiques en sortie des voteurs même en présence de défaillance ce qui ne pose pas de problème si chacun définit sa version, seul la première émise comptera).

4.5 Scénarios de défaillance (1.5 point)

Indiquez les différents scénarios de défaillance que vous pouvez tolérer (plusieurs défaillances simultanées possibles), ceux que vous pouvez détecter mais non tolérer, et ceux que vous ne pouvez pas détecter.

4.6 Architecture de réplication (0.5 point)

Une telle architecture est elle utile :

a) pour améliorer la fiabilité des résultats produits si l'on suppose une probabilité de défaillance de 0.5 par réplique logicielle par exécution (comparez vous au cas où l'on utiliserait une seule réplique).

b) pour améliorer la disponibilité du résultat du vote si l'on suppose une probabilité que le matériel défaille sur une période T de 0.5 (T durée d'une exécution de la fonction f). (Comparez vous au cas où le voteur est déployé sur un seul calculateur)

Répondez en justifiant pour chaque cas.