SYSTÈMES TEMPS RÉEL EMBARQUÉS CRITIQUES

MASTER COMASIC / MASTER SETI / 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 Ordonnancement Mono-Coeur (5 points)

On considère le système suivant de tâches synchrones à échéances implicites. Tout d'abord, on ne s'intéresse qu'aux paramètres de budget C et de de période T. On considère le système ordonnancé par Rate Monotonic.

	C	T	S1	S2	S3	B
T1	1	6	2	0	0	
T2	1	6	0	1	0	
T3	2	8	0	0	2	
T4	1	12	3	3	1	
T5	3	12	1	2	1	

1.1 Calcul d'ordonnancement (3 points)

Calculer le taux d'utilisation et conclure sur l'ordonnancabilité. $5\times(2^{\frac{1}{5}}-1)=0.74$

On souhaite vérifier autrement l'ordonnançabilité du système, notamment en calculant les temps de réponse. Expliquer comment simplifier cette vérification en tirant partie du fait que plusieurs tâches ont la même période. Faire les calculs nécessaires et conclure sur la capacité de RM à ordonnancer ce système.

1.2 Ordonnancement et synchronisation (2 points)

On considère désormais que les tâches partagent des sémaphores et dans la table ci dessus, on indique la durée de la section critique d'une tâche par verrou. On utilise le protocole de synchronisation PIP. Rappeler en quoi il consiste et donner son impact en terme de nombre de blocages des tâches lorsqu'elles partagent plusieurs sémaphores non-imbriqués. Expliquer comment calculer les temps de blocage des tâches et effectuer ce calcul pour l'exemple ci dessus. Justifier le calcul (qui bloque qui et pourquoi).

2 Ordonnancement Partitionné (3 points – COMASIC uniquement)

On cherche à calculer le taux d'utilisation limite des algorithmes d'ordonnancement pour systèmes **partitionnés de M multi-coeurs**. De sorte que si le taux d'utilisation d'un système est inférieur à cette limite, il est ordonnançable par un algorithme d'ordonnancement pour systèmes partitionnés. Nous allons construire un ensemble de tâches défavorable pour ces algorithmes puis en déduire le taux d'utilisation limite.

2.1 Ordonnancement partitionné pour multi-coeurs (1 point)

Rappeler la définition et les principes d'un ordonnancement partitionné pour multi-coeurs.

2.2 Spécification des tâches (1 point)

On définit un ensemble de N tâches identiques de taux d'utilisation U. Donner une propriété sur U pour que deux tâches ne puissent être co-localisées sur un seul cœur.

2.3 Taux d'utilisation limite (1 point)

Calculer le nombre de tâches nécessaires pour que le système ne soit pas ordonnançable et en déduire le taux d'utilisation limite et du coup, le nombre de processeurs On rappelle que $\lceil x \rceil$ représente le plus petit entier supérieur à x

3 Ordonnancement Multi-Coeurs (3 points)

Dans un système comportant un processeur composé de deux cœurs identiques, les tâches sont exécutées suivant une politique d'ordonnancement globale « global Deadline Monotonic » préemptive qui consiste :

- A tout moment de sélectionner la tâche prête de plus petite échéance Di.
- A autoriser la migration des tâches à tout moment.

Soit l'ensemble de tâches périodiques synchrones suivantes :

Nom	Ci	Di	Pi
T1	**10**	**11**	**20**
T2	**1**	**4**	**5**
T3	**4**	**6**	**6**

3.1 Premier ordonnancement (1 point)

Calculer l'ordonnancement de ces tâches sur les 15 premières unités de temps. Toutes les échéances sont-elles respectées ?

3.2 Second ordonnancement (2 points)

Supposons maintenant que la période de T2 soit égale à 6. Calculez à nouveau l'ordonnancement sur les 15 premières unités de temps. Que constatez-vous maintenant ? Est-ce surprenant ? Expliquez pourquoi.

4 Integrated Modular Avionic (5 points)

4.1 Questions de cours (1 point)

En avionique, l'architecture modulaire intégrée s'appuie sur des systèmes partitionnés ARINC653 assurant l'isolation spatiale et temporelle des applications :

- à quoi correspond l'isolation spatiale et temporelle ?
- Quels sont les principes de fonctionnement d'un ordonnanceur ARINC653 ?

4.2 Configuration de l'ordonnancement (2 points)

Le tableau ci-dessous fournit la liste des tâches du système avec leurs périodes, niveau de criticité, et budget en termes d'occupation de la ressource de calcul. En supposant que les tâches sont regroupées dans des partitions ARINC653 en fonction de leur niveau de criticité, calculez la configuration de l'ordonnancement ARINC653. Répondez en représentant cette configuration sur un chronogramme.

Tâche	Période	Criticité	Budget
T1	100 ms	A	5.00%
T2	200 ms	A	20.00%

T3	100 ms	B	20.00%
T4	200 ms	B	30.00%
T5	100 ms	C	25.00%

4.3 Délais de communication (2 points)

On suppose désormais qu'à chacune de ses exécutions, la tâche T1 envoie une donnée d à la tâche T5. On suppose que les ports de communications inter-partition sont vidangés toutes les « minor frames » (MIF). Enfin, on considère que la date d'émission de d correspond à la date de fin de la « partition window » dans laquelle s'exécute T1, et la date de réception de d correspond à la date de début de la « partition window » dans laquelle s'exécute T5. En réutilisant la configuration d'ordonnancement que vous avez établie à la question précédente, au bout de combien de temps la tâche T5 reçoit la dernière valeur produite par la tâche T1 ? Justifiez votre réponse et représentez ce délai sur le chronogramme obtenu à la question précédente.

5 Bus et Réseaux Temps Réel (2 points)

Expliquer les inconvénients d'un réseau de type Ethernet dans le cas d'un système avionique critique. Expliquer pourquoi malgré tout, il présente un avantage certain. Expliquer comment Ethernet a été adapté pour les besoins des systèmes avioniques critiques.

WCET (3 points)

Considérez le code ci-dessous pour les questions suivantes:

```
// Initialize sample acquisition
extern void sampleInit(sample_t samples[MAX_SAMPLES]);
// Returns true when a sample is available
extern bool sampleAvailable();
// Reads a sample, returning true on success.
extern bool sampleRead(sample_t *sample);
// Process a samplem returning true on success.
extern bool sampleProcess(sample_t *sample);
// Terminate processing of a batch of samples.
extern void sampleDone(sample_t sample[MAX_SAMPLES], unsigned int nSamples);
// states of state machine
enum state_t { INIT, WAIT_FOR_SAMPLE, READ_SAMPLE, PROCESS_SAMPLE, DONE };
// Acquire at most nSample samples, returning the number of samples acquired.
unsigned int processSamples(unsigned int nSamples) {
  enum state_t state = INIT;
  sample_t samples[MAX_SAMPLES];
  unsigned int sampleIndex = 0;
  unsigned int iteration = 0;
  while (state != DONE) {
    // check if we should abort
    if (++iteration > MAX_ITERATION) return 0;
    switch (state) {
      case INIT:                                                    // BB6
        sampleInit(samples);
        state = WAIT_FOR_SAMPLE;
        break;
      case WAIT_FOR_SAMPLE:                                         // BB5
        if (sampleAvailable()) state = READ_SAMPLE;
        else state = WAIT_FOR_SAMPLE;
        break;
      case READ_SAMPLE:                                             // BB4
        if (sampleRead(&samples[sampleIndex])) state = WAIT_FOR_SAMPLE;
        else state = PROCESS_SAMPLE;
        break;
      case PROCESS_SAMPLE:                                          // BB7
        if (sampleProcess(&samples[sampleIndex])) {
          sampleIndex++;
          if (sampleIndex > nSamples) state = DONE;
          else state = WAIT_FOR_SAMPLE;
        }
        else state = DONE;
        break;
      case DONE:                                                    // BB8
        sampleDone(samples, sampleIndex);
        break;
    }
  }
  return sampleIndex;
}
```

5.1 Bornes des boucles (1 point)

Considérez la fonction processSamples ci-dessus. Donnez une formule symbolique qui borne le nombre maximal d'itérations exécutées par la boucle **while** quand cette fonction est appelée. Vous pouvez utiliser les noms des variables dans votre formule. Est-il possible de donner une telle borne en utilisant la variable nSamples ? Votre réponse doit être justifiée.

5.2 Implicit Path Enumeration (IPET) (1 point)

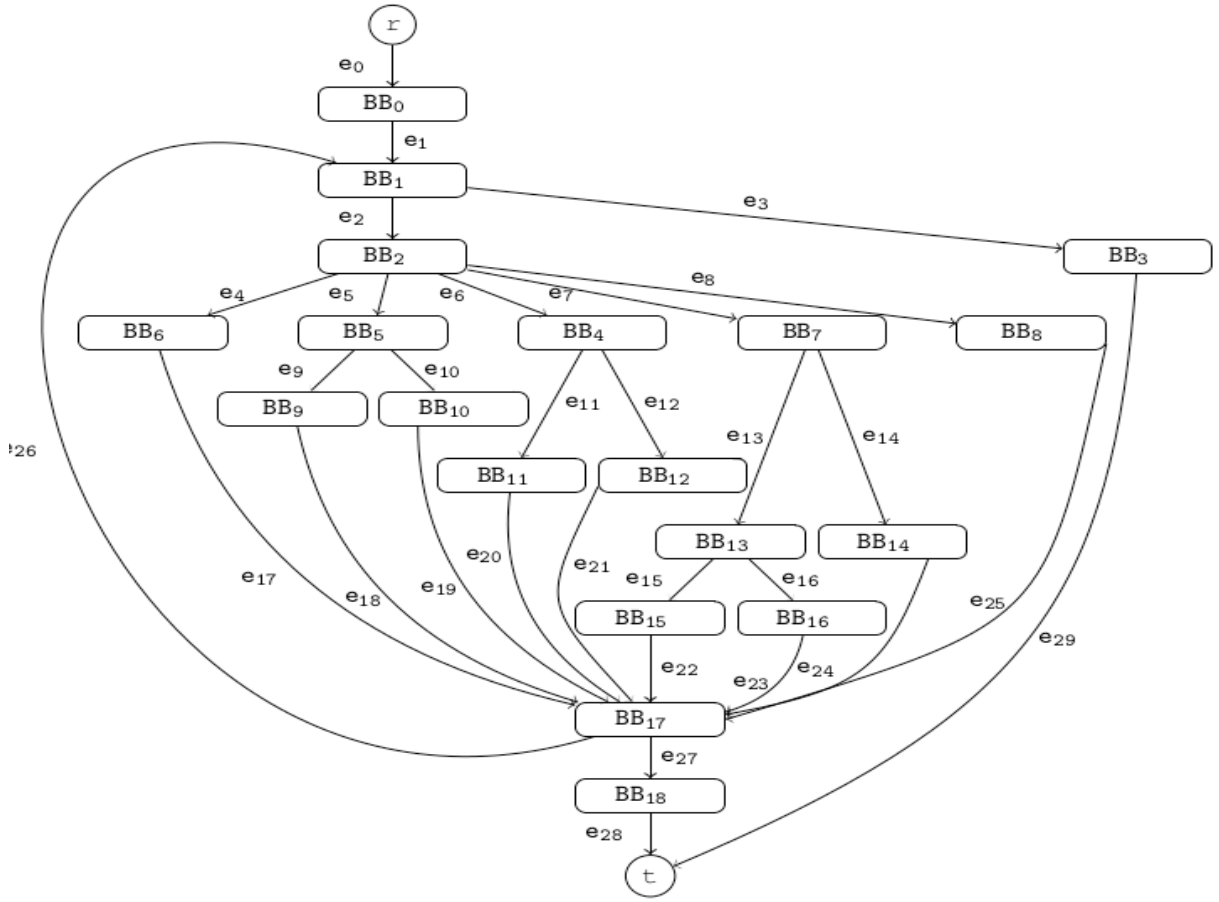
Expliquez l'approche Implicit Path Enumeration (IPET), qui est basée sur la programmation linéaire. Quels sont les différents types d'équations construites pour IPET. Nommez-en au moins trois. Quelles sont leurs fonctions ? Quelle est la fonction des variables qui apparaissent dans ces équations ? Quel est la fonction de ces variables par rapport de la fonction d'objective ?

5.3 Flot du programme sous IPET (1 point)

Vous trouverez plus bas un sous-ensemble des équations générées par IPET pour le code du programme et le graphe de flot de contrôle ci-dessous. Classez les équations par rapport aux (au moins) trois types d'équations que vous avez identifiés pour la question précédente.

Précisions : Les variables MAX_ITERATION et MAX_SAMPLES sont constantes. Elle peuvent donc apparaître comme des symboles dans les équations. Les noms des variables de flot (flow variables) des blocs de base et des arcs du graphe de flot de contrôle sont disponibles dans la figure suivante. Les commentaires situés à droite du code, vous permettent de associer les lignes de code aux blocs de base correspondants.

- $BB_8 = e_1$
- $BB_{17} = e_{17} + e_{18} + e_{19} + e_{20} + e_{21} + e_{22} + e_{23} + e_{24} + e_{25}$
- $e_{28} + e_{29} = 1$
- $BB_4 \leq MAX\_SAMPLES \bullet e_1$
- $e_{26} \leq MAX\_ITERATION \bullet e_1$



6 Cache (3 points – SETI uniquement)

6.1 Politique d'écriture (1 point)

Expliquez les difficultés associées avec la politique d'écriture **write-back** durant l'analyse statique du contenu du cache.

6.2 Analyse statique basé sur l'âge (1 point)

Une technique d'analyse du cache classique est l'interprétation abstraite basée sur l'âge de bloc de mémoire (memory block). Quelle est la différence entre les deux analyses **Must** et **May** ? L'age fourni par chacune de ces analyses vous permet de classer certains types d'accès à la mémoire comme **Always-Hit** ou **Always-Miss**. Expliquez les liens entre la classification et l'analyse **Must**/**May.**

6.3 Persistance (1 point)

L'analyse statique basée sur l'âge vous permet de classer des accès à la mémoire en trois catégories (a) **Always-Hit**, (b) **Always-Miss**, et (c) **Not-Classified**. Pourquoi les accès **Not-Classified** sont-ils problématiques ? Quelle amélioration apporte le concept de persistance ? Y-a-t-il un bloc de base du code ci-dessus (Question) qui est susceptible de profiter de cette technique d'analyse ?