25/11/2022 :

Application :

Tâche	r	C	D	T
T_1	0	inconnu	70	100
T_2	0	17	35	60
T_3	0	11	40	40

① Vérifier la condition nécessaire d'ordonnançabilité

② Tester différents algo d'ordonnancement et la condition suffisante pour RMA et EDF

③ Etablir le chronogramme en Round Robin avec $Z = 40$

① Condition nécessaire $\sum_i \frac{C_i}{T_i} \leq 1 \iff \frac{X}{100} + \frac{17}{60} + \frac{11}{40} \leq 1$

$\iff \frac{X}{100} \leq 1 - \frac{17}{60} - \frac{11}{40}$

$\iff X \leq \left(1 - \frac{17}{60} - \frac{11}{40}\right) \cdot 100$

$\iff X \leq 44$

② . RMA :

Condition suffisante $U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n\left(2^{\frac{1}{n}} - 1\right)$ avec n : nombre de tâches

$\iff \frac{X}{100} + \frac{17}{60} + \frac{11}{40} \leq 0,779$ avec $n = 3$

$\iff X \leq 100 \cdot \left(0,779 - \frac{17}{60} + \frac{11}{40}\right)$

$\iff X \leq 22$

. EDF :

Condition suffisante $U = \sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1$

$\iff \frac{X}{70} + \frac{17}{35} + \frac{11}{40} \leq 1 \iff X \leq 70 \cdot \left(1 - \frac{17}{35} - \frac{11}{40}\right)$

≤ 16

Remarque: On ne peut pas conclure sur l'ordonnançabilité si la condition suffisante est violée.
En revanche, il faut respecter la condition nécessaire dans tous les cas !

Compilation du code avec temps d'exécution: godbolt.org

Par exemple choix d'un compilateur RISC-V rv32gc clang
↳ Reduced Instruction Set

Affiche le langage machine associé.

En assembleur, la précision i à la fin d'une instruction montre que la valeur à manipuler est directement comprise dans l'instruction (ex addi sp, sp, 32 pour ajouter 32 au contenu du registre sp et à stocker dans sp)

On peut compter le nombre d'instructions par ligne de programme :

Pour un if() on compte le nombre d'instructions dans les 2 choix et on peut considérer le choix le plus désavantageux
le plus probable en fonctionnement nominal de l'application.

Dans notre exemple, on trouve X = 18 ou 24

L'application est ordonnançable

On ne peut peut être pas en EDF et on peut faire un RMA.

Préciser les niveaux d'optimisation avec les options -O -O1 -O2 -O3 à la compilation.

↳ Il est conseillé de ne pas dépasser O2 sinon il peut y avoir des approximations

(3) RR 10: Code sur cliffe L. Biffle

Supposons qu'il y a 100 € sur le compte, que quelqu'un souhaite retirer 100 € (→ T2), qu'une autre personne souhaite déposer 100 € → T2 et une tierce personne consulte le solde → T3

Tâche 1 : condition remplie donc BRANCH pas ORTS LBB2. 2

|

après 10 tops d'horloge

↓

Tâche 2: a0 = 200 registre du solde du compte

↓

Tâche 3: ret 100 à la consultation du compte

↓

22 T2 : Écriture - 100 € sur le compte

T2 : a stoché 2000 Tâche T2 finie

T3 : ret 100 T3 finie

T1 : retourne 1 => retrait validé T1 finie

On a mit en lumière un problème d'accès à une ressource partagée :

On a retirer -100 € alors que l'on a consulté un solde de 100€ et que l'on pense avoir déposé 200 €.

Solutions : 1) Mettre en place un verrou std::mutex M;

2) Compiler avec une optimisation -O2

Préparation TP2 : Free RTOS

↳ freertos.org

Consulter Tasks and Co-routines.

Vocabulaire de Scheduling Basics

Fonctions dans API Reference > Task Creation

✗ Nom_Fonction() -> renvoie un pointeur

✓ Nom_Fonction() -> renvoie du void

↳ paramètre usStackDepth = taille de la mémoire utilisée

> TaskDelay pour effectuer une attente vTaskDelay (1000 / portTICK) attente de 1s normalisée au nombre de TICS d'horloge

> vTaskSuspend pour suspendre en passant le xHandle en paramètre

Ces fonctions constituent les appels système => Kernel de notre application

ESP-IDF de Espressif pour les outils de l'OS (flasher sur un µC, compiler, débuger)

↳ Extension VScode existante

↳ exemple sur le GitHub : Get Started