1 Introduction

- super-loop model : boucle while(1)
- event-driven model : interruption
- RTOS RealTime Operating System

2 Scheduling

- Périodique (time-driven)
- Apériodique (event-driven)
- Sporadique (tâches faites au moins tous les T_{max})
- Arrière-plan (download)

2.1 Static scheduling

Fixer lors de l'écriture du code. Impossible de la modifier lors de exécution du code

2.1.1 Paramètres des tâches

- P_x : période de la tâche
- D_x : deadline
- C_x : cas le plus long de exécution de la tâche
- $C_x < P_x$: cette condition doit toujours être vrai
- $D_x = P_x$: simplification

2.1.2 Marche à suivre (super-loop model)

- Définir le plus petit temps d'exécution commun aux tâches
- Est ce possible de construire une table en prenant compte des deadline (si possible le faire)
- Exécuter la table des temps sur un cycle
- Les tâches peuvent être découpées en sous parties (ou ralentie (wait))

2.1.3 Avantages

- Facile à implémenter (après avoir trouver la table)
- Prévisible et facile à debuger

2.1.4 Désavantages

- Difficile à trouver la time table (peux être très longue est découpée)
- Exécution des tâches fixes
- Le délais entre deux appelle de tâche est fixe (1cycle complet)
- Le programme test tout le temps chaque device (polling)
- Cette méthode s'adapte mal (modification compliqué)

2.2 Event-driven scheduling

Basé sur des évènements généré par une source et utilisé par le receveur

Les évènements peuvent arriver de manière prévisible (cyclique) ou non (bouton)

Meilleur que la super-loop (pas de polling)(rapidité de gestion des entrée bouton)

2.2.1 Avantages

- Efficace (pas de polling)
- Rapide (mécanisme hardware) interruption bouton
- Modifiable (on peut rajouter facilement des ISR)

2.2.2 interruption

priorité, masquage, imbrication des ISR, allocation de mémoire modifiable

Opération atomique dans les interruptions, pas d'allocation dynamique, ni d'utilisation de mutex

2.3 Dynamic scheduling

Basé sur la priorité des tâches ou durée au choix. L'exécution des tâches va changer durant l'exécution du code

Une tâche ne peux pas être coupée (RTC Run To Completion)

2.3.1 Marche à suivre

- Si pas de tâche prête, il reste en IDLE
- SI pas de tâche en cours, il démarre la tâche READY la plus prioritaire
- Lors de l'exécution d'une tâche pas de préemption (RTC)
- Une fois fini l'exécution la tâche passe en WAITING jusqu'à qu'elle repasse en READY

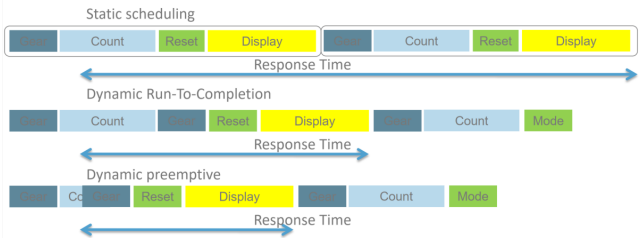
2.4 Dynamic preemptive scheduling

Une tâche peut être coupé par une autre tâche plus prioritaire

2.4.1 Marche à suivre

- Une activité de tâche peut être mise en attente (bloquée)
- Une tâche qui attend ne revient pas au CPU. Elle doit être signalée par une ISR ou une autre tâche
- Seul le scheduler déplace les tâches entre ready et running
- Sinon : round-robin entre les tâches de même priorités

2.5 Comparaison des temps de réponse



- preemption offre de meilleurs temps de réponse
- preemption nécessite plus de mémoire et plus complexe

- preemption introduit des vulnérabilité (mutex déjà utilisé)
- introduit le concept de RTOS

2.6 Comparaison des performances

- Difficile voir impossible de faire des comparaisons définies
- Il faut à la fois des analyses théoriques et des mesures empiriques

  Analyse théorique : théorie des files, modélisation

  Mesures empiriques : avec de la simulation ou des mesures sur des systèmes existants
- Mesures de performances

  Temps d'attente : temps perdu par les tâches à attendre lorsqu'elles sont ready

  Temps turnaround : temps pris de la soumission de la tâche à sa complétion

  Utilisation du CPU

  Throughput : nombre de tâches exécutée par unités de temps

  Temps de réponse

  Equitabilité entre tâches

3 Multi tâche

Besoin de découper le code en petit élement

- code lisible, structuré et documenté
- code testable et modulaire
- développement réutilise du code existant pour réduire le temps de développement
- code peut être découper entre plusieurs personnes
- code malléable et facile à mettre à jour

Organisation des codes sous formes de classes et méthodes. Le c++ est donc parfait pour ceci.

Il y a deux types de tâches

- event-triggered : se lance lors d'évènement
- time-triggered : cyclique

3.1 multitâche et OS

C'est un OS qui nous permet de faire du multitâche

3.2 Tâches vs Processus

Process :

- possède des ressources privée
- un processus peut contenir plusieurs tâches
- nécessite plusieurs cœur (vrai //)

Task :

- un seul cœur (faux //)
- ressource partagée entre deux taches possibles (gestion avec mutex ou sémaphore)
- s'exécute à tour de rôle selon leurs priorités

3.3 RTOS

Le temps réel garanti la complétion d'un process dans un intervalle de temps défini. Par contre il ne dit pas quel temps ça doit prendre, ça doit juste être toujours prédictible.

Un RTOS est un OS qui :

- Sert des applications temps-réelles
- Répond à des requêtes dans une délai garantit et prédictible
- Process les données dans un nombre déterministe de cycles

Un RTOS est plus pour des performances déterministiques, plutôt que un high troughput

- Les performances peuvent être mesurées par latence (temps qu'il prend)
- Les performances peuvent être mesurées par gigue (variation du temps qu'il prend)
- Soft RTOS : plus de gigue, généralement répond aux deadlines
- Hard RTOS : moins de gigue, répond aux deadlines de manière déterministe

3.4 Tâches et RTOS

C'est l'OS qui détermine quelle section du programme va être exécuter et l'accès aux ressources Il gère aussi la synchronisation entre les tâches.

3.5 Mbed OS

3.5.1 Task scheduling

mbed utilise RTX5 pour ARM avec une faible latence

Les priorités sont configurées telles qu'aucune préemption apparait entre ces handler

C'est une combinaison de priorité et d'un scheduling de round-robin

- Round Robin pour les tâches de même priorités
- Basé sur la priorité pour les autres tâches

3.5.2 Scheduling Options

- Preemptive : chaque tâche à une priorité différente
- Round-Robin : toutes les tâches ont la même priorité
- Co-operative multi-tasking : toutes les tâches ont la même priorité et chaque tâche va jusqu'à un appelle bloquant

Par défaut c'est Round-Robin préemptif

3.5.3 Context switching

Le Thread Control Block rend le changement de contexte plus facile

Cela prend du temps (200-300 cycles)

- save state, load other state
- change state

3.5.4 Thread States

- Running
- Ready
- Waiting/Blocked
- Inactive/Terminated

3.5.5 Thread synchronization

Dans un système multi-tâches, les différentes tâches peuvent tenter d'obtenir la même ressource ou peuvent attendre l'apparition de différents évènements

Dans certains cas, un tâche donnée peut entrer dans un état Waiting ou Blocked

Il y a plusieurs Waiting states

3.6 Evènements

Les évents sont utiles pour attendre certaines conditions

- Un thread attends pour une condition spécifique
- La condition peut être faite d'un ou plusieurs flags (AND/OR)
- Attente avec timeout est possible
- Un autre thread set cette condition spécifique
- Pas d'attente active

3.7 Mutex

Protection des ressources partagées

- Ils s'assurent qu'un seul thread peut avoir accès à une section de code critique
- On doit être attentif au deadlock ou starvation
- Faire attention quand on a plusieurs mutex
- Toujours redonner le mutex après utilisation

3.8 Deadlock

Condition pour un deadlock (si toutes vraies)

- Le Mutex est inévitable
- Préemption est inévitable
- Prévenir les conditions circulaire d'attente

Si deadlock alors reset (watchdog)

3.9 Sémaphore

Producteur / Consommateur

Un sémaphore gère l'accès des threads à un set de ressources partagées d'un certain type

- A la différence des mutex, un sémaphore peut contrôler l'accès à plusieurs ressources partagées
- Une sémaphore active l'accès et la gestion d'un groupe de périphériques identiques

3.10 Queue/mail

- Queue : FIFO
- Mail : comme une Queue avec de la mémoire pour des messages

3.11 Inversion de priorité

Les tâches les plus hautes bloquent les tâches plus basses cela modifie le temps de réponse

Pour empêcher cela il faut suivre les Resources Access Protocols

3.11.1 Principales sources de d'inversion de priorité

- section non-préemptive
- ressources partagées
- synchronisation et exclusion mutuel

3.12 Resources Access Protocols

- Inversion de priorité est bornée
- Evite les deadlock
- Evite les blocages non-nécessaire
- Calcul facile du temps max de blocage
- Nombre de lbocage max
- Facile à implémenter

3.12.1 Types

- Non preemption (NPP) : haute pri si lock mutex ok
- Priotity Inhertiance (PIP) : si bloque tâche plus haute pri alors prend sa pri (MBED)
- High Locker (HLP) : prend la pri de du mutex. La pri du mutex est definit par la plus haute qui l'utilise
- Priority Ceiling (PCP) : PIP et HLP prend le plus haut

3.12.2 Résumé

	NPP	PIP	HLP	PCP
Inv priorité	yes	yes	yes	yes
Evite deadlock	yes	no	yes	yes
Evite blocage	no	yes	yes/no	yes
calcul max blocage	easy	hard	easy	easy

4 Mémoire

- Code ou texte

  Instructions binaires à être exécutée

  Elle est habituellement read-only

  Le Program Counter (PC) pointe sur la prochaine instruction à être exécutée
- Données statiques: Variables globales/constantes/statiques partagées entre tâches/threads
- Heap :Dynamique allouée avec malloc/free (C) ou new/delete (C++)

- Stack

  Utilisé pour exécuter du code, méthode/fonction appel et retour

  Position est dans le stack Pointer

4.1 Compiler Stages

- Pre-processing

  Remplace les macros qui sont définie par un # dans le code

  Fusionne les sous-fichiers (.c/.cpp, .h) en un seul fichier (remplace les includes .h par son texte interne)

- Parser (analyseur)

  Lit le code C

  Vérifie les erreurs de syntaxes

  Créer un code intermédiaire (représentation en arbre)

- Optimisateur Haut-niveau : modifie le code intermédiaire (indépendant du processeur)

- Générateur de code (dès qu'on connait l'architecture)

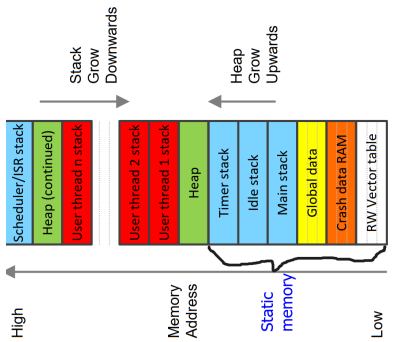
  Crée le code assembleur étapes par étapes pour chaque noeud du code intermédiaire

  Alloue les registres à différentes utilisations

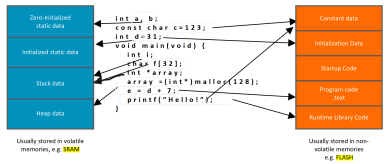
- Optimisateur Bas-niveau : modifie le code assembleur (les parties sont spécifiques au processeur)

- Linker/Loader : créer une image exécutable du fichier objet

4.2 Model mémoire



4.3 Emplacement des données



4.4 Types de données

- const : fixe en ROM
- volatile : modifiable ISR pas d'optimisation
- static : garde leurs valeurs entre deux appelles

4.5 Hiérarchie mémoire (plus au moins rapide)

- registres : dans le CPU
- cache : (static RAM)
- main memory : dynamic RAM, volatile data
- secondary memory : flash/hard disk
- tertiary memory : tape libraries

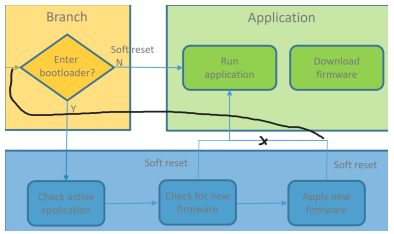
4.6 Memory Protection Unit

Améliore la sécurité peut définir 8 régions avec différents accès

- unité programmable
- gère les accès mémoires et alloue des privilèges
- monteur les transactions (data, instructions)
- active un FAULT_EXCEPTION lors d'une violation d'accès
- (OS) définit régions mémoires
- (OS) définit les permissions pour y accéder

5 Bootloader

- C'est une application (logiciel)
- Permettre à un software/firmware d'être mis à jour
- Sans l'utilisation d'un hardware spécialisé(No Jtag prog)
- Différents protocoles pour recevoir le software update



Vector table, bootloader, application, candidat(nouvelle application)

6 Tests