1 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

1.1 multitâche et OS

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

1.2 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

1.3 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

1.4 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.

1.5 Mbed OS

1.5.1 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

1.5.2 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

1.5.3 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

1.5.4 Thread States

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

1.5.5 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

1.6 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

1.7 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

1.8 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)

1.9 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

1.10 Queue/mail

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

1.11 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

1.11.1 Principales sources de d'inversion de priorité

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

1.12 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

1.12.1 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

1.12.2 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