1 Scheduling

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

1.1 Static scheduling

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

1.1.1 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

1.1.2 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))

1.1.3 Avantages

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

1.1.4 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)

1.2 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)

1.2.1 Avantages

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

1.2.2 interruption

priorité, masquage, imbrication des ISR, allocation de mémoire modifiable
Opération atomique dans les interruptions, pas d'allocation dynamique, ni d'utilisation de mutex

1.3 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)

1.3.1 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

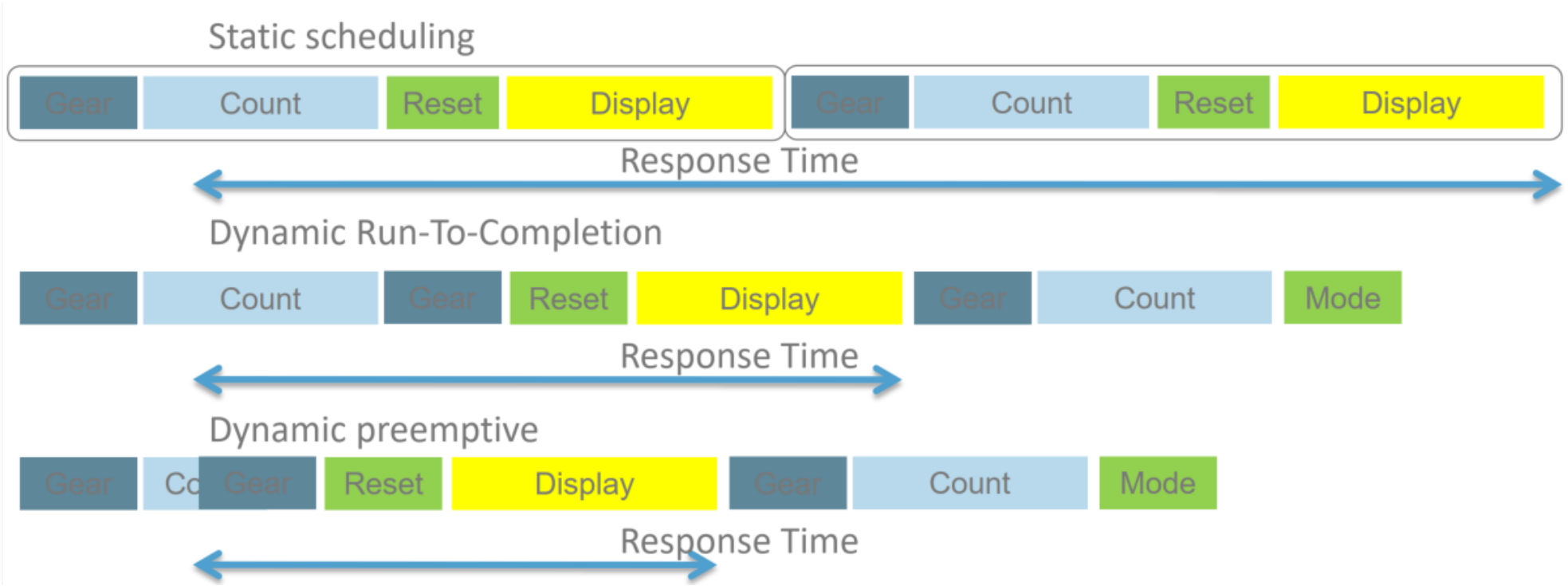
1.4 Dynamic preemptive scheduling

Une tâche peut être coupé par une autre tâche plus prioritaire

1.4.1 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

1.5 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

1.6 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