Systèmes d'exploitation embarqués et temps-réel

Chapitre 2 - suite : Ordonnancement temps-réel

Aimen Bouchhima

2021-2020

Earliest Deadline First (EDF)

- Ordonnanceur à priorité variable
- A tout instant, La priorité de la tâche est inversement proportionnelle à son échéance absolue
  - Échéance plus proche = priorité plus élevée

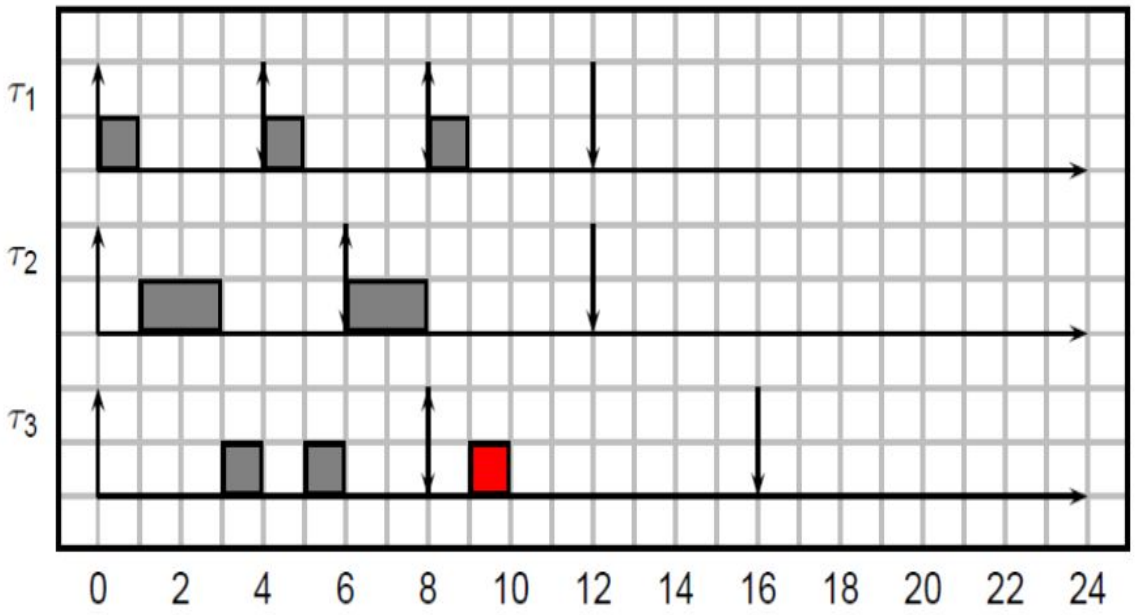
Exemple (EDF)

Tâche	Ci	Ti
$\tau 1$	4	8
$\tau 2$	6	12

- Utilisation processeur ?
- Diagramme d'ordonnancement sur l'intervalle d'analyse ?

RM vs EDF

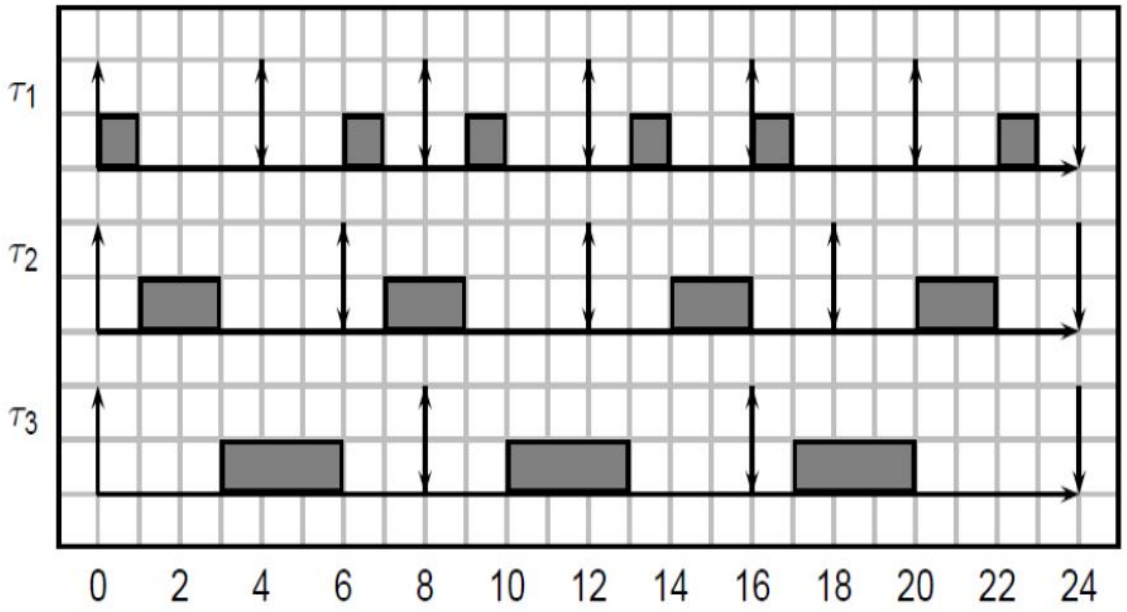
Tâche	Ci	Ti
$\tau 1$	1	4
$\tau 2$	2	6
$\tau 3$	3	8



RM

RM vs EDF

Tâche	Ci	Ti
$\tau 1$	1	4
$\tau 2$	2	6
$\tau 3$	3	8



EDF

RM vs EDF : Conclusion

- L’ordonnancement RM est la politique d'ordonnancement à priorité **fixe** optimale
  - Aucune autre politique à priorité fixe ne peut faire mieux
  - Pourtant, elle pourrait ne pas atteindre 100% d'utilisation du processeur
- EDF est la politique d'ordonnancement à priorité **dynamique** optimale
  - Tant que l’utilisation du processeur est inférieur ou égale à 100%, le système sera ordonnançable

RM vs EDF : Conclusion

- En pratique, dans les systèmes embarqués temps-réel, on préfère RM plutôt que EDF
  - RM est plus facile à implémenter
    - Ordonnanceur plus simple → moins de ressources consommées
  - RM est plus clair et robuste
    - Plus facile à comprendre ce qui se passe en cas de problèmes (ex: surcharge)
    - Si une tâche consomme plus que son temps prévu (WCET), les tâches de plus fortes priorités ne seront pas touchées

Partage de ressources et temps-réel

Partage de ressource et temps-réel

- Souvent, les tâches temps-réel ont besoin d'interagir
- La forme la plus facile d'interaction est via le mécanisme de la mémoire partagée
  - Nécessité de protéger l'accès à la mémoire partagée
    - Exclusion mutuelle
- Quel impact sur l'ordonnancement temps-réel ?

Condition de concurrence

- Le problème de la condition de concurrence (Race Condition) se produit lorsque deux tâches accèdent en lecture et écriture à un même emplacement mémoire (ex variable globale)
- Exemple

initialement **counter** = 5

Thread 1

```
...
counter ++
...

R1 = load (counter);
R1 = R1 + 1;
counter = store (R1);
```

Thread 2

```
...
counter --
...

R2 = load (counter);
R2 = R1 - 1;
counter = store (R2);
```

Condition de concurrence

- Le problème de la condition de concurrence (Race Condition) se produit lorsque deux tâches accèdent en lecture et écriture à un même emplacement mémoire (ex variable globale)
- Exemple

initialement **counter** = 5

Thread 1	Thread 2
...	...
counter ++	counter --
...	...

Condition de concurrence

```
R1 = load (counter);
R1 = R1 + 1;
counter = store (R1);
R2 = load (counter);
R2 = R2 - 1;
counter = store (R2);
```

counter = 5

```
R1 = load (counter);
R1 = R1 + 1;
R2 = load (counter);
R2 = R2 - 1;
counter = store (R1);
counter = store (R2);
```

counter = 4

```
R1 = load (counter);
R1 = R1 + 1;
R2 = load (counter);
R2 = R2 - 1;
counter = store (R2);
counter = store (R1);
```

counter = 6

- Le résultat dépend de l'ordre d'exécution
- Comment prévenir ce problème ?

Exclusion mutuelle

- Autoriser un seul thread à entrer dans une section critique
  - Section critique : partie du code où la tâche accède à la ressource partagée
- Le verrouillage (lock) est un mécanisme d'implémentation de l'exclusion mutuelle

Protection de la section critique

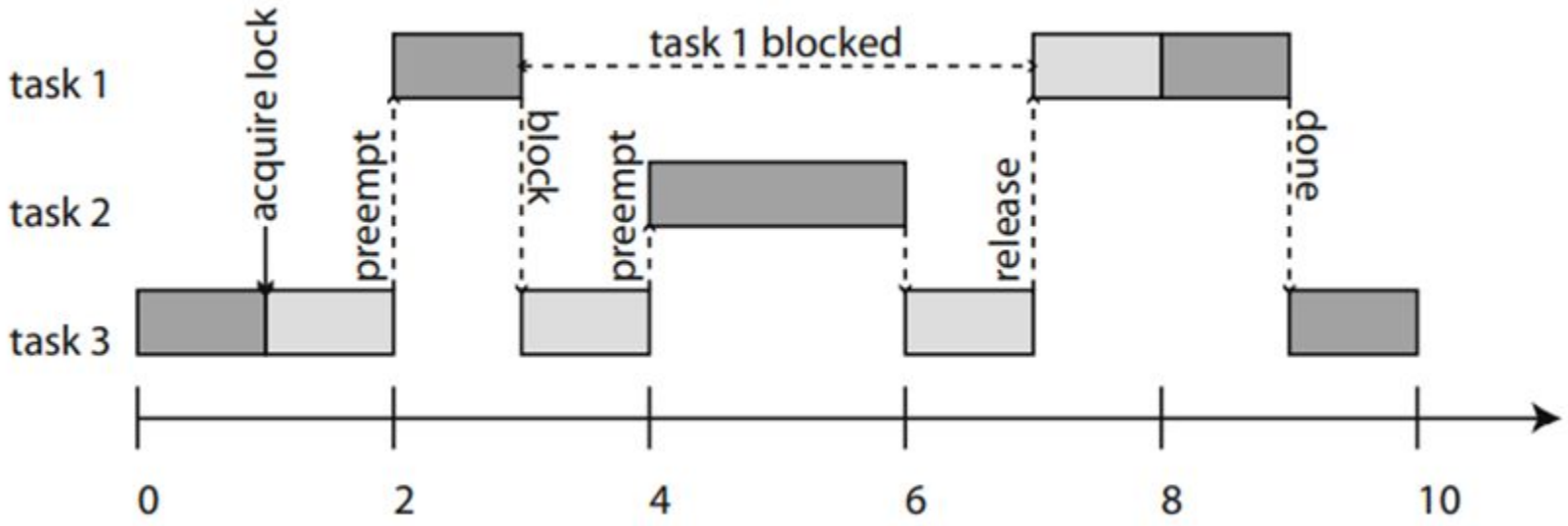
- Solution générale
  - Protéger la section critique via un verrou (lock)
  - Acquérir le verrou en entrant, le libérer en sortant
  - le verrou peut être un mutex, un sémaphore, ...

```
do {
     acquire lock;
     critical section
     release lock;
     remainder section
} while(TRUE);
```

Le problème de l'inversion de priorité

- Le problème derrière l'échec de la mission pathfinder sur mars en 1997
- Une tâche de plus faible priorité bloque une autre tâche de plus forte priorité

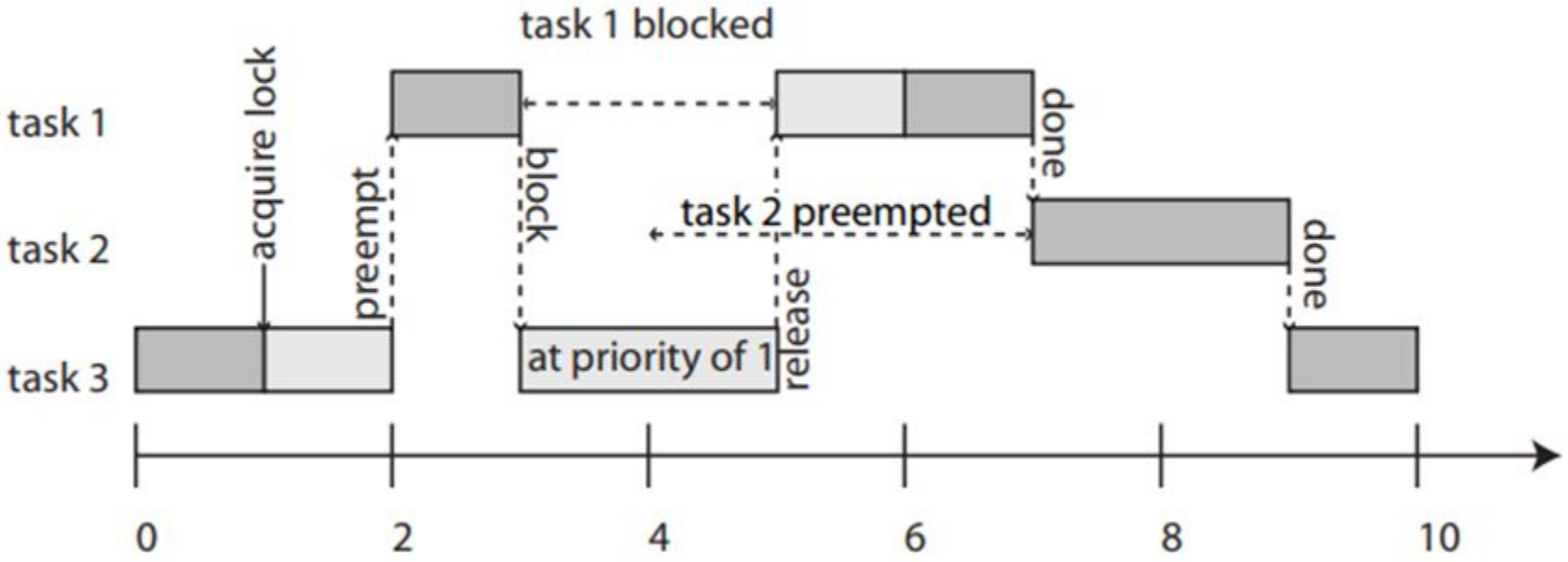
Illustration de l’inversion de priorité



Solution 1 : Héritage de priorité

- La tâche qui détient le verrou hérite la priorité de la tâche qui essaye d’acquérir le même verrou
- Conséquence : empêche l'inversion de priorité
  - La tâche qui détient le verrou ne peut pas être préemptée par une tâche de priorité intermédiaire

Solution 1 : Héritage de priorité



Solution 1 : Héritage de priorité

- Cependant : n'empêche pas l'inter-blocage (deadlock)